Containment of Nested XML Queries

Xin Dong Alon Y. Halevy Igor Tatarinov

{lunadong,alon,igor}@cs.washington.edu
University of Washington, Seattle

Abstract

Query containment is the most fundamental relationship between a pair of database queries: a query Q is said to be contained in a query Q' if the answer for Q is always a subset of the answer for Q', independent of the current state of the database. Query containment is an important problem in a wide variety of data management applications, including verification of integrity constraints, reasoning about contents of data sources in data integration, semantic caching, verification of knowledge bases, determining queries independent of updates, and most recently, in query reformulation for peer data management systems. Query containment has been studied extensively in the relational context and for XPath queries, but not for XML queries with nesting.

We consider the theoretical aspects of the problem of query containment for XML queries with nesting. We begin by considering conjunctive XML queries (c-XQueries), and show that containment is in polynomial time if we restrict the fanout (number of sibling sub-blocks) to be 1. We prove that for arbitrary fanout, containment is coNP-hard already for queries with nesting depth 2, even if the query does not include variables in the return clauses. We then show that for queries with fixed nesting depth, containment is coNP-complete.

Next, we establish the computational complexity of query containment for several practical extensions of c-XQueries, including queries with union and arithmetic comparisons, and queries where the XPath expressions may include descendant edges and negation. Finally, we describe a few heuristics for speeding up query containment checking in practice by exploiting properties of the queries and the underlying schema.

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

1 Introduction

Query containment is a fundamental relationship between a pair of database queries: in the relational context, a query Q is said to be contained in a query Q' if the answer for Q is always a subset of the answer for Q', independent of the current state of the database. In the context of XML queries with nesting (or more generally, queries over complex objects), where answers are trees, we require the answer of Q be embedded in the answer of Q'. This paper considers the formal aspects of determining query containment for XML queries with nesting. We begin by describing the many motivations for studying query containment.

1.1 Motivation

Originally, query containment was studied for optimization of relational queries [9, 33]. Removing redundant parts of a query reduces the number of joins performed by the query processor. Determining that a minimized query is equivalent to the original one requires a containment test.

More recently, query containment has found several applications in systems that need to reason about contents of data sources, such as data integration and peer-data management systems. As an illustration, consider the problem of query answering in a peer-data management system (PDMS) [21, 35, 32, 23, 4].

A PDMS offers a decentralized architecture for sharing data among *peers*, removing the need for a mediated schema that is required in data integration systems. Each peer has an associated schema that represents its domain of interest. In addition, a peer can contribute actual data. Semantic relationships between peers are described locally by mappings between pairs (or small sets) of peers. Note that a PDMS offers a data-sharing architecture that is a strict generalization of data integra-

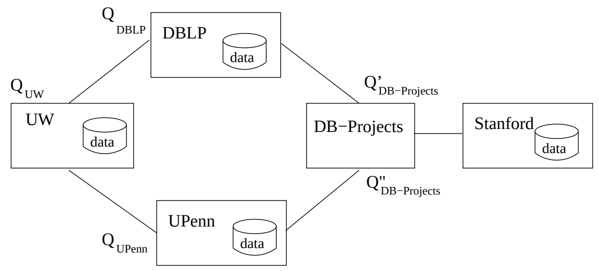


Figure 1: A PDMS for the database research domain. Query containment is needed in order to reformulate a query on a peer onto its neighbor (e.g., from UW to DBLP), and to prune redundant reformulations (e.g., one of the two paths from UW to DB-Projects may result in a redundant query.)

tion systems [36, 8, 18]. Figure 1 shows a PDMS for supporting sharing of database-research related data.

The semantic mappings between peer schemas enable *reformulating* a query over a peer to queries over its neighbors. Given a query at a peer, the query processor applies reformulation iteratively to explore all possible semantic paths in the PDMS, until it reaches every relevant peer. In Figure 1, the user has posed a query over the UW peer. The PDMS has reformulated the input query over the UPenn, DBLP, and DB-Projects peers. This is the first place where query containment is needed: query reformulation algorithms, typically based on algorithms for answering queries using views [20, 25, 21], require checking query containment.

Continuing with the example, because there are two paths from UW to DB-Projects, two reformulated queries over DB-Projects have been obtained: $Q'_{DB-Projects}$ and $Q"_{DB-Projects}$. Next, the PDMS will reformulate these two queries over the Stanford peer. However, one of these two queries may be redundant in the sense that all the answers it produces are guaranteed to be produced by the other query. In recent experiments on the XML-based Piazza PDMS [35], we showed that pruning such redundant queries significantly speeds up query reformulation in a PDMS. Detecting that a reformulated query is redundant reduces to the problem of query containment.

Furthermore, it turns out that it is also crucial to minimize reformulated queries after every reformulation step [35, 16]. As already described, query minimization involves query containment.

Query containment is also important for *semantic caching*, which is a current need in several recent data integration products [14, 1]. Intuitively, checking whether an answer to a new query is already in the cache amounts to a containment check between the new query and the cache.

It is important to note that XML is increasingly being used in the kinds of middleware and data-sharing applications mentioned above. Therefore it is important to develop query containment techniques for XML queries. Finally, we note that query containment has also been used in maintenance of integrity constraints [19, 15] and knowledge-base verification [26].

1.2 Related Work

Query containment has been studied in depth for the relational model, beginning with conjunctive queries [9, 2], then acyclic queries [38], queries with union [33], negation [27], arithmetic comparisons [24, 37, 27, 39, 19], recursive queries [34, 10] and queries over bags [11, 22].

Query containment for XML poses two challenges: the use of XPath expressions to specify patterns on the input data, and the nesting structure of the resulting tree. Several recent works considered query containment for XPath in isolation. In [30] it is shown that for a simple fragment of XPath that contains descendant axis(//), wildcards(*), and qualifiers (or branching, denoted [...]), but without either tag variables or disjunctions, query containment is coNP-complete. If we drop any one of the constructs *, //, and [...] in the above case, query containment is in PTIME [3, 31]. In [12] the authors studied XPath containment under a limited use of tag variables and equality testing, and showed the problem is Π_2^p-complete in general and NP-complete if no disjunction or wildcards are allowed. Finally, [17] showed that containment of queries with regular path expressions on general cyclic graphs is PSPACE-hard. In [35] we describe a practically-motivated algorithm for containment of XQuery queries with nesting, but that algorithm is complete only for queries *without* nesting.

Containment of queries returning nested structures has been considered for only the general case of queries over complex objects [28]. The study shows that the containment problem can be reduced to the problem of *query simulation*; however, the proposed reduction is not accurate. As we demonstrate, the algorithm of [28] considers only a *subset* of the simulations that a sound algorithm should check. Furthermore, from a practical perspective, we consider several extensions not addressed in [28], and we show how the complexity depends on restrictions such as the nesting depth and fanout in the queries.

1.3 Example

Example 1.1: Figure 2 shows an example of two XQuery queries, where containment cannot be determined solely based on comparing their respective

```
Q: for $group in /group
   where $group/gname/text() = "database"
   return
   < area > {
      for $person in $group/person
      return
      < person >
         < name > {$person/text()} < /name >
         {for $paper in $group/paper
         where $paper/author/text() = $person/text()
         return
         < paper > {$paper/title/text()} < /paper > }
      < /person > }
   < /area >
```

```
Q': for $group in /group
   return
   < area > {
      for $person in $group/person
      return
      < person >
         < name > {$person/text()} < /name >
         < group > {$group/gname/text()} < /group >
         {for $paper in $group/paper
         where $paper/author/text() = $person/text()
         return
         < paper > {$paper/title/text()} < /paper > }
      < /person > }
   < /area >
```

Figure 2: The query on the left, Q, is contained in the query on the right, Q', but not the other way around. We note that checking XPath containment on this example is not sufficient to establish nested XML query containment.

XPath components. The two queries, Q and Q', take an input document consisting of paper and person elements where a person element contains a person name and a paper element contains a title and author subelements. In the output, the person elements have a name and paper subelements instead. Clearly, checking containment of XPath fragments in every block is not sufficient to establish that Q is contained in Q'. In addition, one must consider the structure of the queries, the predicates and returned values that appear in each block and how the XPath expressions are spread across the query blocks.

1.4 Our Contributions

We begin by considering a fragment of XML queries, called *conjunctive XML Queries (c-XQueries)*, which covers many queries used in practice (analogous to select-project-join queries in SQL). We show that query containment for this fragment can be checked in polynomial time if we restrict the fanout (number of sibling sub-blocks) in the query to be 1. However, if we allow arbitrary fanout, then query containment is coNP-hard even for queries with nesting depth 2 and even if the query does not include variables in the return clauses. Since XPath expressions in c-XQueries can be modeled as acyclic conjunctive queries, and containment of unnested acyclic queries is in PTIME, our result isolates the exact effect of nesting on the complexity of query containment.

Next, we show that query containment for c-XQueries with arbitrary fanout but *fixed* nesting depth is coNP-complete. Our technique is based on considering a finite number of *canonical databases* (a technique also used in [24, 17]). Here, the appropriate set of canonical databases is obtained by inspecting a set of *canonical answers* to the query, each representing a possible structure for the answer tree. We note that without restricting the nesting depth of the query, the number of canonical databases that need to be inspected can be super-exponential, and the exact complexity for this case remains open.

Last, we consider several extensions of c-XQueries that are important in practice. In particular, we consider queries with union, negation, and queries where the XPath expressions may include descendant edges (besides wildcards and branching). In each of these cases we show that even with fanout 1, query containment is coNP-complete, and that query containment for queries with fixed nesting depth is still coNP-complete. We also consider nested queries with equality predicates on tag variables, and show that containment is NP-complete for queries with fanout 1 and Π_2^p-complete for queries with arbitrary fanout but fixed nesting depth. Then, we show that for queries with arithmetic comparisons on tag variables, containment is Π_2^p-complete both for queries with fanout 1 and for queries with arbitrary fanout but fixed nesting depth. Finally, we describe a few heuristics for speeding up query containment checking in practice by exploiting properties such as cardinality knowledge gleaned from the query and the schema of the underlying XML data. In particular, these heuristics significantly reduce the number of canonical databases that actually need to be considered.

We note that there are two aspects of XQueries that we do not consider: bag semantics and order. Our results entail necessary conditions for query containment in these cases – if containment does not hold for sets, it certainly does not hold for ordered lists or bags. Although order is part of the semantics of XQuery, it is less important in data management applications of XML. In fact, query containment with ordered lists has not been addressed even for XPath. Our focus here is on the additional challenges that arise from adding nesting to XML queries.

The paper is organized as follows. Section 2 for-

```
<project>
  <title>Piazza</title>
  <member>Alice</member>
</project>
<project>
  <title>Tukwila</project>
  <member>Bob</member>
</project>
```

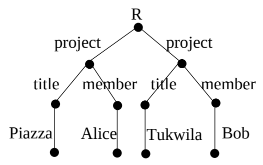


Figure 3: An XML instance and corresponding tree.

mally defines the problem. Section 3 considers cc-XQueries, and Section 4 extends the results to c-XQueries. Section 5 describes our results for extensions of c-XQueries, and Section 6 concludes.

2 Preliminaries

We begin by defining XML instances and the different query language fragments we consider. We then define containment on instances and on queries.

2.1 XML Instances and XML Trees

In our discussion we model XML instances and elements as unordered edge-labeled trees. Nodes in the tree represent XML (sub)elements, and have identifiers from a domain $\mathcal{N}$, which is disjoint from the domain of tag constants, $\mathcal{T}$. Edges between nodes represent nesting relationships, and the labels on the edges (taken from $\mathcal{T}$) represent XML tags. The identifier of the root is $\Re$. Note that in the tree representation, labels on edges leading to leaf nodes correspond to *text values* in the XML document, while internal edge labels correspond to *XML element tags*. We use edge labeling instead of node labeling to distinguish tag and node variables. All of our results also hold in the node-labeled representation.

Example 2.1: Figure 3 shows an XML instance and its corresponding tree that include information about projects in a research group. □

In the rest of the paper, we use the terms *XML instance* and *XML tree* interchangeably.

2.2 Conjunctive XML Queries

We start by considering a subset of XML queries, called *conjunctive XML queries (c-XQueries)*. c-XQueries are similar in spirit to select-project-join queries in SQL, and therefore already form useful subset of XQuery.

Syntax: c-XQueries satisfy the following restrictions:

- In a c-XQuery, the returned variables are bound to tag names or text values only. Note that if an XML query has a variable bound to an XML element, we can easily expand the RETURN clause according to the schema and transform the query to a c-XQuery. Henceforth, the term *tag variable* will refer to a variable that can be bound to either a text value or an element tag (i.e., both types of labels on tree edges).
- XPath expressions in a c-XQuery contain only child axis (/), wildcards (*) and branching ([...]). In Section 5 we extend our results to decide query containment for more general XML queries, where there are descendant axis (//) or negations in XPath expressions, or there are unions or comparisons in XPath expressions or WHERE-clause conditions.
- To exclude disjunction, we require that sibling blocks always return *distinct* tag constants. Consequently, a block can return a tag variable only when it has no siblings. Section 5 discusses containment of disjunctive queries.

```
Q: for $x in /project return
< group > {
  for $s in $x/title/text() return
  < projtitle >{$s}< /projtitle > } {
  for $t in $x/member/text() return
  < name >{$t}< /name > }
< /group >
```

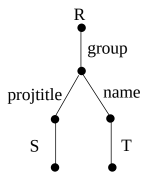


Figure 4: An example c-XQuery and its head tree.

A c-XQuery consists of nested *query blocks*. A query block may have a set of sub-blocks. The *fanout* of a query block is the number of its immediate sub-blocks. A query with no sub-blocks has a *nesting depth* of 1. The nesting depth of a query is 1 plus the maximal nesting depth of its sub-blocks. The nesting depth of the query is the depth of its outer-most block. In the example c-XQuery Q shown in Figure 4, the outer-most query block has fanout 2 and the nesting depth is 3.

The structure of an XML query and its answers can be described using the notion of a query *head tree*. The nodes of the head tree of Q are the query blocks of Q, and there is an edge between the node corresponding to a query block and the node corresponding to its parent block. The label of the incoming edge of a node n in the head tree is the returned tag of the block corresponding to n in Q (which can also be a variable). Note that we consider an expression $\{\$s\}$ a query block with empty for and where clauses and with a return tag of $\$s$; such nodes will appear as leaves of the head tree. Figure 4 also shows the head tree of the example c-XQuery. Note that a head tree is also an XML instance if its variables are substituted with actual values.

For our analysis in Section 3 we define a smaller fragment of c-XQueries, called *constant conjunctive XML queries (cc-XQueries).* A cc-XQuery is a c-XQuery that does not return tag variables. The head tree of a cc-XQuery has constant labels only.

Semantics: The semantics of a c-XQuery is an extension of the semantics of an un-nested conjunctive query. Specifically, each node n in the answer is *generated* by a query block with the same depth as n. Note that since c-XQuery does not allow disjunction, each node has a unique generator. For every valid variable substitution in a query block, we generate an output element with the corresponding tag. When there is at least one satisfying substitution, we evaluate the block's sub-blocks. Note that a variable substitution of a sub-block is an extension of that for its parent block. The output element of the outer-most block of an c-XQuery is the answer to the query.

We note that a c-XQuery can be evaluated on an input XML instance in polynomial data complexity and exponential query complexity.

2.3 Containment of Instances and Queries

An XML tree is a special case of a complex object, where each record is binary. Hence, we follow the definition of containment given in [28]. Specifically, we base XML instance containment on tree homomorphism (not necessarily injective). Following [30], we define an *embedding* as follows:

Definition 2.2 (Tree embedding). *Given two trees, a node mapping ψ from t_1 to t_2 is said to be an* embedding *from t_1 to t_2 if*

- *ψ maps the root of t_1 to the root of t_2,*
- *if node n_2 is a child of node n_1 in t_1, then $\psi(n_2)$ is a child of $\psi(n_1)$, and the edge between n_1 and n_2 has the same label as the edge between $\psi(n_1)$ and $\psi(n_2)$.* □

Definition 2.3 (XML Instance Containment). *Let e and e' be two XML instances. e is* contained in *e', denoted as $e \sqsubseteq e'$, if the tree of e can be embedded in the tree of e'.* □

This definition of XML tree containment has several desirable properties. First, containment is reflexive and transitive. Second, it is the smallest order relation for XML trees which is a congruence, i.e., $e_1 \sqsubseteq e'_1 \wedge \cdots \wedge e_n \sqsubseteq e'_n$ implies $<t>e_1, \ldots, e_n</t> \sqsubseteq <t>e'_1, \ldots, e'_n</t>$ where $<t>e_1, \ldots, e_n</t>$ is an XML instance constructed by adding a common parent $<t>$ element over $e_1, \ldots, e_n$. Third, our containment definition is consistent with set semantics. An XML instance with multiple copies of its subelement is contained in the XML instance that has just one copy of each subelement. From a theoretical perspective, there are additional justifications for this definition. This notion of containment has also been used previously for partial information [7] and or-sets [29], and it coincides with the simulation relation between complex objects represented as graphs [5, 6].

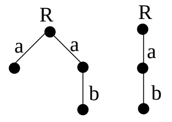


Figure 5: Two XML instances that contain each other but are not equivalent.

Note that this definition of containment is not antisymmetric: $e \sqsubseteq e'$ and $e' \sqsubseteq e$ do not imply $e = e'$. As an example, consider the two XML instances in Figure 5. They contain each other, but are not equivalent.

The following sections make use of *minimal XML instances* that we define as follows:

Definition 2.4 (Minimal XML Instance). *An XML instance e is said to be minimal if the XML tree of e does not contain a pair of sibling subtrees (sub-instances) e' and e'' anywhere in e such that $e' \sqsubseteq e''$.* □

Based on the definition of XML tree containment, we define XML query containment as follows:

Definition 2.5 (XML Query Containment). *Let Q and Q' be two XML queries. Q is* contained *in Q', denoted as $Q \sqsubseteq Q'$, if for every input XML instance D, $Q(D) \sqsubseteq Q'(D)$.* □

3 Containment of cc-XQueries

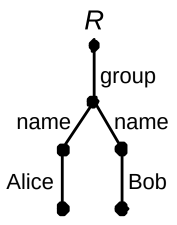
We begin by considering query containment for cc-XQueries. A cc-XQuery does not return tag variables, and thus can be viewed as a generalization of a boolean conjunctive query (i.e, a conjunctive query with an empty head). Nevertheless, unlike boolean conjunctive queries, cc-XQueries can return one of *several* tree structures, rather than only true or false. Although cc-XQueries are rarely useful in practice, we study them for two reasons: first, they already show some of the important lower bounds on query containment; and second, they help us obtain insights on the techniques we use to establish containment, which later carry over to c-XQueries.

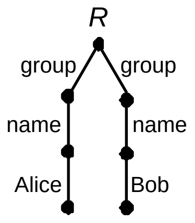
Intuitively, given a pair of queries Q and Q', we cannot check that for *every* possible XML input D, $Q(D) \sqsubseteq Q'(D)$. Hence, our goal is to find a finite set of representative inputs, called *canonical databases*, which have the property that $Q \sqsubseteq Q'$ if and only

```
Q:for $x in /project return
  < group > {
    for $y in /project/member
    return
    < name > {
      where $y = “Alice”
      return < Alice/ >
      where $y = “Bob”
      return < Bob/ > }
    < /name > }
  < /group >

Q’:for $x in /project return
  < group > {
    for $y in $x/member return
    < name > {
      where $y = “Alice”
      return < Alice/ >
      where $y = “Bob”
      return < Bob/ > }
    < /name > }
  < /group >
```

(a)





(b)

Figure 6: Example 3.1: (a) Q and Q'; (b) the answers to Q and Q' on the input XML instance in Figure 3.

if $Q(DB) \sqsubseteq Q'(DB)$ for every canonical database DB.

Our approach is based on considering the different *canonical answers* that can be generated for Q, and creating a canonical database for each canonical answer. One could conjecture that it suffices to consider all the answers corresponding to *prefix subtrees* (the subtrees that contain the root) of Q's head tree. However, the following example refutes this conjecture.[1]

Example 3.1: Consider the two cc-XQueries, Q and Q', in Figure 6(a). The query Q checks whether Alice and Bob are in the research group, and groups them together regardless of their projects. The query Q' also checks whether Alice and Bob are in the research group, but in contrast, groups them according to whether they are working on the same project. Figure 6(b) shows the results of Q and Q' on the XML instance D of Example 2.1. $Q(D) \not\sqsubseteq Q'(D)$, and thus $Q \not\sqsubseteq Q'$. In contrast, containment *does* hold for canonical databases generated for all prefix subtrees of the head tree. □

3.1 Canonical Answers and Databases

The observation leading to our first result is that it suffices to consider canonical answers that are minimal XML instances and are *contained* in the head tree (which is different from being prefix subtrees of the head tree).

[1]Furthermore, since the result in [28] considers only these subtrees, this example entails that the algorithm of [28] offers only a necessary condition for query containment, but not a sufficient one.

Definition 3.2 (Canonical Answer of a cc-XQuery). *Let Q be a cc-XQuery and H be its head tree. A* canonical answer *of Q is a minimal XML instance CA, such that $CA \sqsubseteq H$.* □

For each canonical answer we define a canonical database as follows.

Definition 3.3 (Canonical Database of a cc-XQuery). *Let Q be a cc-XQuery, and CA be a canonical answer of Q. Q's* canonical database *for CA, denoted as DB_{CA}, is an XML instance, s.t. for each node N of CA where N's generator query block is $\hat{q}_n$, the following holds: Let $p_0/p_1/\ldots/p_n$ be a path expression in $\hat{q}_n$, where p_0 is an optional node variable from an ancestor query block. For each $p_i, i \in [1, n]$, there is a distinct node, labeled p_i, that is a child of the node for p_{i-1}. If p_0 is absent, then p_1 is a child of DB_{CA}'s root.* □

The number of canonical databases for a cc-XQuery is the same as the number of canonical answers. The size of a canonical database is polynomial in the size of its corresponding canonical answer.

For example, Figure 7(a) shows six canonical answers for Q in Example 3.1. Figure 7(b) shows the corresponding canonical databases. (Note that tag names are abbreviated.)

3.2 Query Containment Algorithm

Our first result shows that to test query containment, it suffices to consider only canonical databases constructed from the canonical answers. In the following theorem, DB_{CA} (DB'_{CA}) refers to the canonical database of $Q(Q')$ corresponding to the canonical answer CA. The following theorem gives several equivalent characterizations of query containment. For complete proofs of the theorems in this paper, see [13].

Theorem 3.4 (Containment of cc-XQueries). *Let Q and Q' be two cc-XQueries. The following three conditions are equivalent:*

1. *$Q \sqsubseteq Q'$;*
2. *for every canonical database DB of Q, $Q(DB) \sqsubseteq Q'(DB)$;*
3. *for every canonical answer CA of Q, (a) CA is a canonical answer of Q'; and (b) $DB'_{CA} \sqsubseteq DB_{CA}$.* □

The proof of the above theorem is based on two important properties of canonical answers and canonical databases.

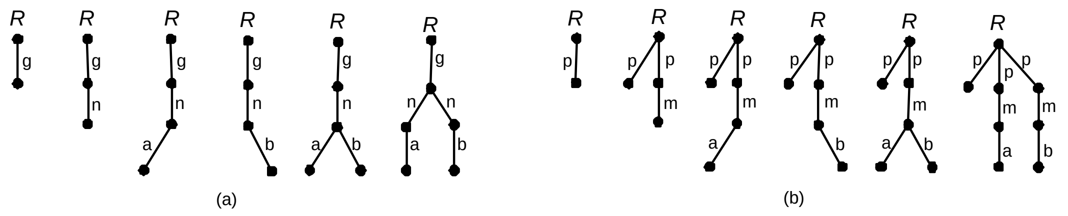


Figure 7: Example 3.1: (a) Q's canonical answers; (b) Q's canonical databases.

Lemma 3.5. *Let Q be a cc-XQuery and D be an XML instance. There exists a unique canonical answer CA of Q, such that $Q(D) \sqsubseteq CA$ and $CA \sqsubseteq Q(D)$.* $\square$

Lemma 3.6. *Let Q be a cc-XQuery, CA be a canonical answer of Q, DB_{CA} be the canonical database for CA of Q, and D be an XML instance. $CA \sqsubseteq Q(D)$ if and only if $DB_{CA} \sqsubseteq D$.* $\square$

Proof sketch for Theorem 3.4:
(1) $\Rightarrow$ (2): Follows from the definition.

(2) $\Rightarrow$ (3): Consider a canonical answer CA and its canonical database, DB_{CA}. According to Lemma 3.6, $CA \sqsubseteq Q(DB_{CA})$. Since condition (2) holds, $Q(DB_{CA}) \sqsubseteq Q'(DB_{CA})$. Putting the above two containments together, we have $CA \sqsubseteq Q'(DB_{CA})$. This implies that (a) holds. Applying Lemma 3.6 again gives $DB'_{CA} \sqsubseteq DB_{CA}$. Hence, (b) holds.

(3) $\Rightarrow$ (1): To show $Q \sqsubseteq Q'$, we need to show for every XML instance D, $Q(D) \sqsubseteq Q'(D)$. According to Lemma 3.5, there exists a unique canonical answer CA of Q, such that $Q(D) \sqsubseteq CA$ and $CA \sqsubseteq Q(D)$. According to Lemma 3.6, $DB_{CA} \sqsubseteq D$. Since conditions (a) and (b) hold, $DB'_{CA} \sqsubseteq DB_{CA}$. So $DB'_{CA} \sqsubseteq D$. Applying Lemma 3.6 again gives $CA \sqsubseteq Q'(D)$. Based on containment transitivity, $Q(D) \sqsubseteq Q'(D)$. $\square$

The third condition of Theorem 3.4 is of practical importance, and will become useful in the complexity analysis. The condition states that we do not actually have to evaluate the two queries on each of the canonical databases. Instead, it suffices to check tree embedding on each pair of canonical databases, which can be done in polynomial time in the size of the canonical databases.

However, as the following analysis shows, the number and sizes of canonical databases, determined by the number and sizes of canonical answers, are quite large. Let m be the largest fanout of a query block in Q, and let d be the nesting depth of Q. We can show that the maximal size of Q's canonical answers is $\Theta(m \cdot 2^{m\cdot\left(2^{\cdot^{\cdot^{m\cdot(2^m)}}}\right)})$ (which is a tower of exponents with $d-1$ levels), while the number of canonical answers is similar, but d levels tall. However, this is only a temporary setback. We will show that there is no need to consider so many canonical answers.

3.3 Effect of the Fanout

In this section, we show that the fanout of the queries has a significant impact on the complexity of query containment. A cc-XQuery is said to be *linear* if the fanout of each query block is at most 1. We show the following:

Theorem 3.7 (Containment for linear cc-XQueries). *Let Q and Q' be cc-XQueries. If either of them is linear, testing $Q \sqsubseteq Q'$ is in PTIME.* $\square$

We note that this is the only case in which nesting does not add to the complexity of containment in comparison to similar queries without nesting. As we will soon see, the restriction on the fanout is crucial for obtaining polynomial-time complexity.

Proof sketch: The proof of Theorem 3.7 is based on the following observations. First, to check that $Q \sqsubseteq Q'$, where Q is a linear cc-XQuery, the number of canonical answers we need to consider is equal to the nesting depth of Q, denoted as d, and the sizes of canonical answers are bounded by d. Second, as entailed by the third part of Theorem 3.4, for each such canonical answer we need to perform an embedding test on the corresponding canonical databases, which can be done in polynomial time in the sizes of the canonical databases, bounded by the sizes of the queries. Specifically, the time complexity is $O(d \cdot |Q| \cdot |Q'|)$. $\square$

The following theorem shows that the restriction on the fanout is critical. The following lower bound is proved by a reduction from the complement of 3SAT.

Theorem 3.8 (coNP-Hardness). *Testing containment of cc-XQueries with nesting depth 2 and arbitrary fanout is coNP-hard.* $\square$

3.4 cc-XQueries with Fixed Nesting Depth

In this section we show that for cc-XQueries with *any* fixed nesting depth, query containment is in coNP. Hence, we obtain the following result:

Theorem 3.9 (coNP-Completeness). *Let Q and Q' be cc-XQueries. If either of them has a fixed nesting depth, testing $Q \sqsubseteq Q'$ is coNP-complete.* $\square$

The key observation behind Theorem 3.9 is to further reduce the number of canonical answers (and hence of canonical databases) we need to consider. As we show below, it suffices to consider *kernel canonical answers*.

Definition 3.10 (Kernel Canonical Answer). *Let Q be a cc-XQuery. Let d be the nesting depth of Q and c be the maximum number of path steps in a query block of Q. A canonical answer CA of Q is called a* kernel canonical answer *if the following hold: (1) the root node has a single child, and (2) suppose N is a node in CA and p is a path from N to a leaf; at most $cd - 1$ siblings of N are roots of paths that are the same as p.* $\square$

In the following lemma, DB_{KCA} (DB'_{KCA}) refers to the canonical database of $Q(Q')$ corresponding to the canonical answer KCA.

Lemma 3.11. *Let Q and Q' be two cc-XQueries. The containment $Q \sqsubseteq Q'$ holds if and only if for each kernel canonical answer KCA of Q, (1) KCA is a canonical answer of Q'; and (2) $DB'_{KCA} \sqsubseteq DB_{KCA}$.* $\square$

Proof sketch: Let CA be the canonical answer with the minimal size that violates $DB'_{CA} \sqsubseteq DB_{CA}$. We show CA must be a kernel canonical answer. First, the root of CA must has a single child. Otherwise, we split it and obtain a set of subtrees, each of which is a canonical answer. At least one of them also violates $DB'_{CA} \sqsubseteq DB_{CA}$, contradicting that CA has the minimal size.

Second, for every node in CA, each conjunct in its generator query block introduces no more than one distinct node to the canonical database. Thus, the size of DB_{CA} is bounded by the size of CA times the maximum number of conjuncts in a query block of Q. On the other hand, considering that CA is minimal, DB_{CA} needs to be at least a certain size in order to guarantee that $DB'_{CA} \not\sqsubseteq DB_{CA}$, and for all canonical answers with a smaller size, denoted as $\widetilde{CA}$, $DB'_{\widetilde{CA}} \sqsubseteq DB_{\widetilde{CA}}$. This size of DB_{CA} is determined by the fanout of CA. Considering the upper bound and the minimum requisite for the size of DB_{CA}, we show that the fanout of CA is bounded, and CA is a kernel canonical answer. $\square$

Now we examine the time complexity of the algorithm derived from Theorem 3.4(3), by analyzing the number and sizes of kernel canonical answers. Let m be the maximum fanout, and b be the number of query blocks in Q. In a kernel canonical answer, the fanout of each node is no more than bcd, since there are no more than cd outgoing edges containing a common path pattern, and there are at most b different path patterns in the query. Hence the size of the canonical answer is in $O((bcd)^d)$. Consider a specific node N in the canonical answer. There are no more than m candidate labels for the edge leading to N. So the number of kernel canonical answers is in $O(m^{(bcd)^d})$. Hence, the time complexity of the algorithm is in $O(m^{(bcd)^d})$.

Corollary 3.12. *Testing containment for cc-XQueries with fixed nesting depth is in coNP. Testing containment for cc-XQueries with arbitrary nesting depth is in coNEXPTIME.* $\square$

Proof sketch: Given two cc-XQueries Q and Q', to check $Q \not\sqsubseteq Q'$, we need to guess a kernel canonical answer of Q, denoted as KCA; construct Q' and Q's canonical databases for KCA, denoted as DB' and DB; and check whether $DB' \not\sqsubseteq DB$. When the nesting depth is fixed, the size of a kernel canonical answer is polynomial in the size of Q. Thus, constructing canonical databases and checking containment both take polynomial time. Hence, query containment is in coNP. When the nesting depth is arbitrary, the size of a kernel canonical answer is exponential in the nesting depth, thus query containment is in coNEXPTIME. $\square$

From Corollary 3.12 and Theorem 3.8 we obtain Theorem 3.9.

Finally, we note that the complexity of query containment for cc-XQueries with arbitrary nesting depth remains an open problem.

4 Containment of c-XQueries

We now consider general c-XQueries, which may return tag variables. A tag variable can be set to any value in $\mathcal{T}$, and therefore the number of candidate answers to a given query is infinite. Consequently, the algorithms for cc-XQueries do not apply directly.

There are two key points underlying our algorithm for checking containment of c-XQueries. First, we consider canonical answers that may contain variables. As we will see, the number of such canonical answers is the same as we had in Section 3. Second, we check query containment by applying a more elaborate procedure for each canonical answer. Specifically, we apply a condition called *query simulation* [28] to a pair of *indexed conjunctive queries* that we create for each canonical answer. Since query simulation, albeit more elaborate, is also in polynomial time, we are able to show that the complexity results for c-XQueries are, for the most part, the same as for cc-XQueries.

4.1 Simulation of Indexed Queries

We begin by explaining indexed conjunctive queries and the condition of query simulation, both from [28]. We represent indexed conjunctive queries using a datalog-like notation, as follows.

$$Q(\bar{I}_1; \ldots; \bar{I}_m; \bar{V}) : -X_1 R_1 Y_1, \ldots, X_n R_n Y_n$$

The body of the indexed conjunctive query is similar to that of an ordinary conjunctive query (except that we write XRY instead of $R(X, Y)$), but the head has a set of tuples of *index* variables, $\bar{I}_1, \ldots, \bar{I}_m$, in addition to the head variables $\bar{V}$. An indexed conjunctive query produces a nested structure: the tuples $\bar{V}$ of the answer are grouped first by the index variables $\bar{I}_1$, then by $\bar{I}_2$, etc., and finally by $\bar{I}_m$.

Example 4.1: Consider query Q in Figure 4. Given the head tree shown in the same figure as the canonical answer CA, the indexed conjunctive query for CA is the following (note that XPath tag names are abbreviated):

$$IQ_{CA}(X; W, Y; S, T) : -\Re pX,\ XtW,\ WSV,\ XmY,\ YTZ$$

Note that the last XML instance shown in Figure 7(a), denoted as CA', is also a canonical answer of Q by applying variable isomorphism. The indexed conjunctive query for CA' is the following:

$$IQ_{CA'}(X; Y_1, Y_2; \emptyset) : -\Re pX,\ XmY_1,\ XmY_2,\ Y_1 a Z_1,\ Y_2 b Z_2$$

By applying Theorem 4.4, which we will describe shortly, we can justify that the query Q' in Example 3.1 is contained in the above query Q, but not the other way around. □

Query simulation is a generalization of query containment for indexed conjunctive queries. Simulation reduces to query containment when the queries contain no index variables. Formally, simulation is defined as follows.

Definition 4.2 (Query Simulation). *Let Q and Q' be two indexed conjunctive queries, each with m sets of index variables. We say that Q'* simulates Q to depth m*, denoted by $Q \preceq_m Q'$, if for any database the following holds:*

$$\forall \bar{I}_1.\exists \bar{I}'_1 \ldots \forall \bar{I}_m.\exists \bar{I}'_m.[\forall \bar{V}.\ (Q(\bar{I}_1; \ldots; \bar{I}_m; \bar{V}) \Rightarrow Q'(\bar{I}'_1; \ldots; \bar{I}'_m; \bar{V}'))] \quad \Box$$

In [28] it is shown that query simulation can be checked by establishing a *simulation mapping* between Q and Q'. In our context, where the body of the conjunctive query is acyclic, finding a simulation mapping can be translated into a tree embedding problem, where the sizes of trees are polynomial in the sizes of the queries. Thus, checking simulation is PTIME in the size of the indexed conjunctive queries.

4.2 Query Containment Algorithm

In general, a c-XQuery may have an infinite number of candidate answers, given all the possible substitutions to the tag variables. Recall that a head tree of a c-XQuery may contain tag variables. When creating canonical answers, we treat the variables as constants. Hence, we can still represent all candidate answers with a finite number of canonical answers.

Given a canonical answer CA, we create an indexed conjunctive query, which is a generalization of the canonical database, as defined below.

Definition 4.3 (Indexed conjunctive query for a canonical answer). *Let Q be a c-XQuery and CA be a canonical answer with depth d. Q's* indexed conjunctive query for CA*, denoted as IQ_{CA}, has the form*

$$IQ_{CA}(\bar{I}_1; \ldots; \bar{I}_{d-1}; V) : -X_1 R_1 Y_1, \ldots, X_n R_n Y_n,$$

and is constructed in two steps. Let N be a node of CA on level $k, k \in [1, d]$, and let $\hat{q}_n$ be N's generator query block in Q. First, if there exists any node M, which may be N's ancestor, descendant, or N itself, where the incoming edge of M is labeled by a tag constant c from $\mathcal{T}$, the generator query block of M returns a tag variable T, and T also occurs in $\hat{q}_n$, we substitute T with c in $\hat{q}_n$. Second, we compose IQ_{CA} as follows:

- *If $k < d$, then $\bar{I}_k$ includes every fresh node variable and tag variable of $\hat{q}_n$.*
- *If $k = d$, then $\bar{V}$ includes the returned tag variable in $\hat{q}_n$, if any.*
- *Let $p_0/p_1/\ldots/p_n$ be a path expression in $\hat{q}_n$, where p_0 is an optional node variable from ancestor query blocks. The body of IQ_{CA} contains $X_0 p_1 X_1$, $X_1 p_2 X_2$, $\ldots, X_{n-1} p_n X_n$, where $X_k, k \in [1, n]$ are distinct node variables, X_0 is an inherited node variable for p_0, or $\Re$ if p_0 is absent.*

We can now show how to test query containment for c-XQueries. The following theorem shows that it suffices to check query simulation on pairs of indexed conjunctive queries generated from the canonical answers. In the theorem, IQ_{CA} (IQ'_{CA}) refers to Q's (Q''s) indexed conjunctive query for CA.

Theorem 4.4. *Let Q and Q' be two c-XQueries. The containment $Q \sqsubseteq Q'$ holds if and only if for every canonical answer CA of Q, (1) CA is a canonical answer of Q' (modulo tag variable isomorphism); and (2) $IQ_{CA} \preceq IQ'_{CA}$.* □

Together with the insights into kernel canonical answers from Section 3, we obtain the following complexity results, which show that the introduction of output tag variables does not make the containment problem harder. (Note that the first bullet has a slightly stronger condition than in Theorem 3.7. Here, we require that Q has fanout 1, rather than *either Q or Q'* does.)

Theorem 4.5. *Let Q and Q' be c-XQueries.*

- *If Q has a maximal fanout 1, then query containment is in PTIME.*
- *For arbitrary fanout, if either Q or Q' has fixed nesting depth, then containment is coNP-complete.*
- *Otherwise, containment is in coNEXPTIME.* □

4.3 Containment Checking in Practice

Containment checking for general nested queries is hard; however, as query sizes in practice tend to be relatively small, query containment can be employed in application. Furthermore, we can drastically reduce the number of canonical answers for containment checking by analyzing the cardinality of elements in the query answer. In this section, we illustrate the effectiveness of this technique based on the example query in Section 1. Our discussion here is meant to suggest possible optimizations that we have found promising. The effect of these techniques still needs to be verified by a thorough experimental evaluation. We also note that the techniques do not decrease the upper bound of the computational complexity.

The intuition behind our technique is that given the query structure and the underlying XML database schema, we can infer the cardinality of elements in the query answer. Canonical answers violating the cardinality constraints do not need to be considered by our algorithm. Specifically, we prune the canonical answers for containment checking according to the following three rules. Suppose we are testing whether Q is contained in Q'. Let $\hat{p}$ be a query block in Q and t be its tag name.

1. ($= 1$): If the schema implies that the variables in the FOR clause of $\hat{p}$ will have exactly one binding, then we only need to consider those canonical answers where t occurs *exactly* once within its parent element. This observation also applies if the FOR and WHERE clauses of $\hat{p}$ are empty. Consider the two queries in Figure 2 for example; to test $Q' \sqsubseteq Q$, we only need to check those canonical answers in which every person element contains exactly one name subelement.
2. (≥ 1): A schema can imply that t will occur *at least once* under its parent element. In the above example, if the schema indicates that every group has one or more person subelements, we only need to check those canonical answers in which every area contains at least one person subelement.
3. (≤ 1): If the schema indicates a certain element occurs *at most once* under its parent element, the set of canonical answers can be constrained similarly to the previous case.

Applying all three techniques to the example in Figure 2 demonstrates the effect of pruning. Testing whether $Q' \sqsubseteq Q$ without applying the above techniques requires considering 71 canonical answers. The first rule prunes 68 of them, and the second rule prunes one more, leaving us with only two canonical answers to be considered.

5 Extensions to c-XQuery

The previous section established the basic complexity results on query containment for conjunctive XML queries with nesting. This section discusses several extensions of c-XQueries that occur frequently in applications.

Union and Disjunction: Union can be introduced into XML queries when two sibling query blocks return XML objects with the same tag. This happens when either two sibling query blocks return the same tag constant, or when at least one of the siblings returns a tag variable, which may be instantiated to the same value returned by the other sibling. This form of union does not affect the complexity of the problem.

Disjunctions in the query's XPath expression or WHERE-clause is another way of expressing certain types of unions. This case can be translated into the above cases, but with an exponential blowup in the size of the resulting queries. We refer to queries with disjunctions as *d-XQueries*. We prove the following result.

Theorem 5.1. *Let Q and Q' be d-XQueries. Query containment is coNP-complete in each of the following cases:*

- *Q has nesting depth of 1,*
- *Q has maximal fanout of 1, and*
- *Q has a fixed nesting depth.*

If Q is a c-XQuery, then the complexity results of Theorem 4.5 still apply. □

Table 1: **Complexity Results for Containment of Nested XML Queries**

Nesting Type	cc-XQueries	c-XQueries	With Union	With Negation	With Descendant Edges (//)	With Equi-join on Tags	With Arithmetic Comparisons
Fanout=1 Arbitrary Depth	PTIME	PTIME	coNP complete	coNP complete	coNP complete	NP complete	Π_2^p complete
Arbitrary Fanout Fixed Depth	coNP complete	coNP complete	coNP complete	coNP complete	coNP complete	Π_2^p complete	Π_2^p complete
General	in coNEXPTIME						

Recall that without union, containment is in polynomial time for the first two cases above. This result is analogous to the relational case, where containment for conjunctive queries and containment for queries with unions are both NP-complete, while containment for disjunctive queries (when union can occur anywhere in the query) is Π_2^p complete. The complexity of the first two cases also increases when we consider negation and descendant edges.

Negation: We consider XQueries where predicates in XPath expressions can contain the "not" operator (e.g. *person*[*not paper*]). This type of negation is similar in spirit to "NOT EXISTS" in SQL. We refer to such queries as *c-XQueries*$^\neg$. We show the following result:

Theorem 5.2. *Let Q and Q′ be c-XQueries*$^\neg$*. Query containment is coNP-complete in each of the following cases:*

- *Q has nesting depth of 1,*
- *Q has maximal fanout of 1, and*
- *Q has a fixed nesting depth.* □

Descendant Edges: We refer to c-XQueries in which the XPath expressions contain the descendant axis (//) as *c-XQueries*$^{//}$. Recall that wildcards(*) and branching([...]) are already allowed in c-XQueries. In [30] it is shown that containment of XPath expressions with //,* and [...] is coNP-complete. The following theorem shows that nesting with fixed depth does not increase the complexity of containment.

Theorem 5.3. *Let Q and Q′ be c-XQueries*$^{//}$*. Query containment is coNP-complete in each of the following cases:*

- *Q has maximal fanout of 1, and*
- *Q has a fixed nesting depth.*

If the number of // in Q is fixed, then the complexity results of Theorem 4.5 still apply. □

Equi-join Predicates: We consider equi-join predicates on tag variables. They result in cyclic queries, where simulation mapping becomes NP-complete [28]. We refer to c-XQueries with equi-join predicates on tag variables as *c-XQueries*$^=$.

Theorem 5.4. *Testing containment of c-XQueries*$^=$ *with maximal fanout 1 is NP-complete. Testing containment of c-XQueries*$^=$ *with arbitrary fanout but a fixed nesting depth is* Π_2^p*-complete.* □

Arithmetic Comparisons: We consider arithmetic comparisons on tag variables. We assume the comparison predicates are interpreted over an ordered and dense domain, and we consider the predicates $<$ and $\leq$. We refer to queries with arithmetic comparisons as *c-XQueries*$^\leq$. In [37] it is shown that containment of cyclic queries with arithmetic comparisons is Π_2^p-complete. We show the following.

Theorem 5.5. *Let Q and Q′ be c-XQueries*$^\leq$*. Query containment is* Π_2^p*-complete in each of the following cases:*

- *Q has maximal fanout of 1, and*
- *Q has a fixed nesting depth.* □

6 Conclusions

XML data is being increasingly used in applications that require integration and sharing of multiple data sources. At several levels, these applications need to reason about the relationship between pairs of queries, either for query reformulation, semantic caching, or various optimization methods. Thus far, in the context of XML, query containment has only been considered for queries expressed as XPath expressions. However, in practice, XQueries are common, and they contain multiple levels of nesting. This paper fills this important gap, by establishing the theoretical underpinnings for XML queries with nesting.

Our results are summarized in Table 1. Our main result is that query containment is coNP-complete for queries with bounded nesting depth, under a variety of conditions. If we consider queries with maximal fanout of 1, then we are able to obtain polynomial-time results in some cases. We have also considered several important practical extensions of the basic conjunctive query language. The main open gap in the complexity analysis is the case of queries with arbitrary nesting depth. In future work, we are interested in extending our containment algorithms to algorithms for answering queries using views. In addition, we would like to perform an empirical evaluation of our containment algorithms and to develop optimizations for common cases.

Acknowledgments

We would like to thank Zack Ives, Richard Ladner, Gerome Miklau, Rachel Pottinger, Dan Suciu, and the reviewers of this paper for their insightful comments and suggestions. This work was supported by NSF ITR grant IIS-0205635 and NSF CAREER grant IIS-9985114.

References

[1] Bea liquid data for weblogic. www.bea.com/liquiddata.

[2] A. Aho, Y. Sagiv, and J. D. Ullman. Equivalence of relational expressions. *SIAM Journal of computing*, (8)2:218–246, 1979.

[3] S. Amer-Yahia, S. Cho, L.V.S.Lakshmanan, and D.Srivastava. Minimization of tree pattern queries. In *Proc. of SIGMOD*, 2001.

[4] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing : A vision. In *Proceedings of the WebDB Workshop*, 2002.

[5] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proc. of SIGMOD*, 1996.

[6] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. Adding structures to unstructured data. In *Proc. of ICDT*, 1997.

[7] P. Buneman, A. Ohori, and A. Jung. Using powerdomains to generalize relational databases. *Theoretical Computer Science*, 91:23–55, 1991.

[8] D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In *Proceedings of KR*, 1998.

[9] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. of STOC*, 1977.

[10] S. Chaudhuri and M. Vardi. On the equivalence of recursive and nonrecursive datalog programs. In *Proc. of PODS*, pages 55–66, San Diego, CA., 1992.

[11] S. Chaudhuri and M. Vardi. Optimizing real conjunctive queries. In *Proc. of PODS*, 1993.

[12] A. Deutsch and V. Tannen. Containment and integrity constraints for xpath. In *KRDB*, 2001.

[13] X. Dong, A. Halevy, and I. Tatarinov. Containment of nested xml queries. Technical Report UW-CSE-03-12-05, Univ. of Washington, 2003.

[14] D. Draper, A. Y. Halevy, and D. S. Weld. The nimble integration system. In *Proc. of SIGMOD*, 2001.

[15] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Verifying integrity constraints on web-sites. In *IJCAI*, 1999.

[16] M. Fernandez, W.-C. Tan, and D. Suciu. Silkroute: Trading between relations and xml. In *WWW*, 1999.

[17] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *Proc. of PODS*, Seattle,WA, 1998.

[18] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proceedings of AAAI*, 1999.

[19] A. Gupta, Y. Sagiv, J. D. Ullman, and J. Widom. Constraint checking with partial information. In *Proc. of PODS*, pages 45–55, Minneapolis, Minnesota, 1994.

[20] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4), 2001.

[21] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *Proc. of ICDE*, 2003.

[22] Y. E. Ioannidis and R. Ramakrishnan. Containment of conjunctive queries: Beyond relations as sets. *ACM Transactions on Database Systems*, 20(3):288–324, 1995.

[23] A. Kementsietsidis, M. Arenas, and R. J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proc. of SIGMOD*, 2003.

[24] A. Klug. On conjunctive queries containing in-equalities. *Journal of the ACM*, 35(1):146–160, 1988.

[25] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS*, 2002.

[26] A. Y. Levy and M.-C. Rousset. Verification of knowledge bases using containment checking. In *Proceedings of AAAI*, 1996.

[27] A. Y. Levy and Y. Sagiv. Queries independent of updates. In *Proc. of VLDB*, 1993.

[28] A. Y. Levy and D. Suciu. Deciding containment for queries with complex objects and aggregations. In *Proc. of PODS*, Tucson, Arizona., 1997.

[29] L. Libkin and L. Wong. Semantic representations and query languages for orsets. In *Proc. of PODS*, 1993.

[30] G. Miklau and D. Suciu. Containtment and equivalence for an xpath fragment. In *Proc. of PODS*, 2002.

[31] T. Milo and D. Suciu. Index structures for path expressions. In *Proc. of ICDT*, pages 277–295, 1999.

[32] W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou. Peerdb: A p2p-based system for distributed data sharing. In *ICDE*, Bangalore, India, 2003.

[33] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1980.

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

[35] I. Tatarinov and A. Halevy. Efficient query reformulation in peer data management systems. In *SIGMOD (to appear)*, 2004.

[36] J. D. Ullman. Information integration using logical views. In *ICDT*, 1997.

[37] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *Proc. of PODS*, pages 331–345, San Diego, CA., 1992.

[38] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. of VLDB*, pages 82–94, 1981.

[39] X. Zhang and M. Z. Ozsoyoglu. On efficient reasoning with implication constraints. In *Proc. of DOOD*, 1993.