Containment of Nested XML Queries

Xin Dong Alon Y. Halevy Igor Tatarinov

{lunadong,alon,igor}@cs.washington.edu
Department of Computer Science & Engineering
University of Washington
Seattle, WA 98195

Abstract

We consider the problem of query containment for XML queries with nesting. Given two queries, Q and Q', Q is said to be contained in Q' if Q produces a subset of the answers of Q' for any database instance. Query containment is important for query optimization, verification of integrity constraints, information integration and verification of knowledge bases. Query containment has been studied in the relational context and for XPath queries, but not for XML queries with nesting.

We begin by considering conjunctive XML queries (c-XQueries), and show that containment is in PTIME if we restrict the fanout (number of sibling sub-blocks) to be 1. We show 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.

We consider several extensions to c-XQueries, including 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 query containment for queries with fixed nesting depth is still coNP-complete. Finally, we show that for queries with equalities on tag variables, containment is NP-complete for queries with fanout 1 and Π_2^p-complete for queries with arbitrary fanout but fixed nesting depth; for queries with arithmetic comparisons on tag variables, containment is Π_2^p-complete for both cases.

1 Introduction

We consider the problem of determining query containment for XML queries with nesting. Query containment is the most fundamental relationship between a pair of queries. In the context of relational databases, where answers are sets of tuples, we say that a query Q is contained in the query Q' if, for any given database instance, the answer of Q is a subset of the answer of Q'. 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'.

Query containment is important in several data management tasks. Originally, query containment was studied for optimization of relational queries [10, 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. Later, query containment was used in answering queries using views [21], maintenance of integrity constraints [19, 15], knowledge-base verification [26] and data integration [35, 9, 17]. In fact, in several recent data integration products [14, 1], which offer an XQuery interface to a multitude of data sources, there was a need for certain kinds of *semantic caching* of answers which, in turn, requires algorithms for query containment of XML Queries with nesting.

Our particular interest in this problem arises from query processing in Peer Data Management Systems (PDMS) [22, 20, 32, 24, 4]. A PDMS offers a decentralized architecture for sharing data among *peers*, removing the need for a mediated schema required in data integration systems. Semantic relationships between peers are described locally by mappings between pairs (or small sets) of peers. These semantic mappings enable *reformulating* a query on a peer to queries on 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 recent experiments on the Piazza PDMS, we have shown that detecting redundant reformulation goals reduces reformulation times by an order of magnitude. Detecting this redundancy cannot be based on XPath containment alone, but requires a query containment algorithm for XML queries with nesting.

Query containment has been studied in depth for the relational model, beginning with conjunctive queries [10, 2], then acyclic queries [37], queries with union [33], negation [27], arithmetic comparisons [25, 36, 27, 38, 19], recursive queries [34, 11] and queries over bags [12, 23].

Query containment for XML faces 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 [13] 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, [16] showed that containment of queries with regular path expressions on general cyclic graphs is PSPACE-hard.

Containment for queries returning nested structures has only been considered for the general case of queries over complex objects [28]. However, that paper only presents a necessary condition for containment, whereas our results offer a necessary and sufficient condition. Furthermore, our analysis exploits the special structure of XML instances to obtain a more refined set of complexity results. In particular, we show how the complexity depends on the nesting depth and fanout in the queries.

We begin by considering a fragment of XML queries, called *conjunctive XML Queries (c-XQueries)*, which covers many queries used in practice. We show that query containment for this fragment is in PTIME 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 [25, 16]). 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 obtain our results by first considering queries that do not include tag variables (cc-XQueries), and then extend the technique of query simulation [28] to obtain results for c-XQueries. 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 (even for cc-XQueries), and the exact complexity for this case remains open.

Finally, we consider several extensions of c-XQueries, including 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. Finally, 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.

This paper is organized as follows. Section 2 formally 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.

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

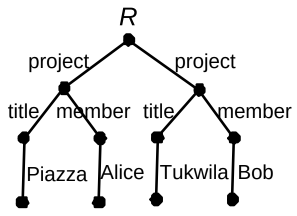


Figure 1: An XML instance and corresponding tree.

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 Objects and Instances

An *XML object* is defined recursively as a binary record: each record contains a *tag* and *contents.* A tag is an atomic value and contents are a set of XML objects. Formally, the definition is the following:

Definition 2.1 (XML Object and Instance). *An* XML object *is a binary record* $[t, \{e_1, \ldots, e_n\}]$*, where* t *is an* atomic tag value *from a* tag constant domain $\mathcal{T}$*, and* $e_1, \ldots, e_n$ *are XML objects. Its abstract syntax is given by*

$$e ::= [\mathcal{T}, \{\epsilon | e_1, e_2, \ldots, e_n\}].$$

An XML instance *is an XML object with a distinguished root tag* $\Re$*.* $\Re$ *appears nowhere else in the instance.* □

In our discussion we model an XML instance as an unordered edge-labeled tree. Nodes in the tree represent XML objects, 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 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 labels on edges leading to internal nodes correspond to *XML element tags.* In our notation, a tag variable can be bound to either kind of label in the tree representation.

Example 2.2: Figure 1 shows an XML instance and its corresponding tree. It lists the projects in a database research group and members in each project. □

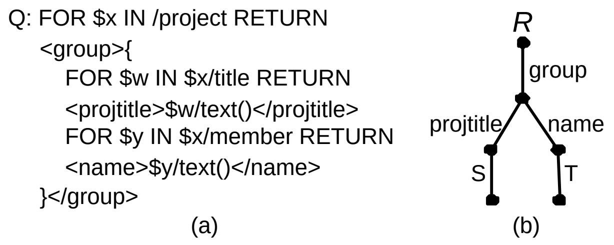
2.2 Conjunctive XML Queries

We now define a fragment of XML queries, called *conjunctive XML queries (c-XQueries).*

Syntax: A c-XQuery represents a FOR-WHERE-RETURN query in XQuery with the following restrictions. First, the returned variables can be bound to only tag names or text values (*i.e.* cannot be bound to XML elements). Second, XPath expressions in c-XQuery contain only child axis (/), wildcards (*) and branching ([...]). Note that the descendant axis (//) is not allowed, nor is negation, union, comparisons except of the form variable=constant (either in XPath expressions or in WHERE-clause conditions). We consider each of these extensions of c-XQuery in Section 5.

For analysis purposes, we often describe c-XQueries with a syntax which is reminiscent of conjunctive queries. The c-XQuery in Figure 2(a) is shown below in our syntax.

Q: group $\leftarrow \Re$ project X {
  projtitle $\leftarrow X$ title W



```
Q: FOR $x IN /project RETURN
   <group>{
     FOR $w IN $x/title RETURN
     <projtitle>$w/text()</projtitle>
     FOR $y IN $x/member RETURN
     <name>$y/text()</name>
   }</group>
              (a)
```

Figure 2: An example c-XQuery: (a) query Q; (b) the head tree of Q.

$\{\mathrm{S} \leftarrow W\mathrm{S}V\ \}$
name $\leftarrow X$ member Y
$\{\mathrm{T} \leftarrow Y\mathrm{T}Z\ \}$
}

A c-XQuery consists of nested *query blocks.* A query block $\hat{q}$ *returns* either a constant in $\mathcal{T}$ or a tag variable (shown on the left-hand side of the $\leftarrow$). We use lowercase letters for tag constants and capital letters for tag variables. In the above query Q, the top-level query block returns a tag constant group, while the leaf query blocks return the tag variables S and T.

In our syntax, path expressions and predicates in the FOR and WHERE clauses are translated into sets of *conjuncts* in the appropriate query blocks. Each conjunct relates two node variables with either a tag constant or a tag variable. We use italic capital letters for node variables, and assume the domain of node variables and tag variables are disjoint. Note that wildcards (*) can be represented by tag variables. A node variable that appears in $\hat{q}$ but not in $\hat{q}$'s ancestor query blocks is called a *fresh node variable* of $\hat{q}$. In the above example, the path expression /project in the top-level query block is translated into the conjunct $\Re$ project X, while the path expression $y/text() is translated into the conjunct $Y\mathrm{T}Z$. The fresh node variables in the last leaf block are Y and Z.

A query block may have a set of sub-blocks. The *fanout* of $\hat{q}$ is the number of sub-blocks it has. A query with no sub-blocks has a *nesting depth* of 1. The nesting depth of $\hat{q}$ 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 above query Q, the outer-most query block has fanout 2 and nesting depth 3.

The answer to the outer-most block of the c-XQuery is the answer to the query (we usually omit the root, which always has the reserved tag $\Re$). We also identify the *head tree* of the query, which is created by removing all the conjuncts from the query (*i.e.* the head tree shows us the shape of the answers.) Note that a head tree is also an XML instance if we apply a substitution to the variables in the head tree. The head tree of the example query Q is shown in Figure 2(b).

To ensure that c-XQueries do not allow 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.

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 contain tag variables (*i.e.* tag variables are not allowed on the left-hand side of the $\leftarrow$, nor as the middle components of conjuncts.)

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. We generate a return tag of a query block for every possible variable substitution that satisfies the conjuncts of the block. 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 the substitution that satisfies its parent block. Finally, in this paper we consider set semantics for queries: at any level of the answer tree, there exists at most one copy of identical sibling sub-trees.

A c-XQuery can be evaluated on an input XML instance in polynomial time of data size, and exponential time of query size. Let $|D|$ be the size of the input instance, $|Q|$ be the size of the query. Each query block is an acyclic query; evaluating it on the input data takes $O(|Q| \cdot |D|)$ time [37]. Let d be the depth of Q, and

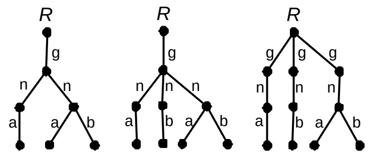


Figure 3: Three XML instances that contain each other pairwise but are not equivalent.

n be the maximum number of fresh variables in a query block of Q. In the worst case, there can be $O(|D|^n)$ number of embeddings for a query block; a query block with depth d can be evaluated for up to $O(|D|^{nd})$ times. This gives the upper bound of c-XQuery evaluation as $O(|Q| \cdot |D|^{|Q|})$.

2.3 Containment and Equivalence

An XML object is a special case of a complex object, where each record is binary. Hence, we follow the definition of containment given in [28].

Definition 2.3 (XML Object Containment). *Let $e = [t, \{e_1, \dots, e_n\}]$ and $e' = [t', \{e'_1, \dots, e'_{n'}\}]$ be two XML objects. e is* contained in *e', denoted as $e \sqsubseteq e'$, if $t = t'$ and $\forall i.\exists i'.e_i \sqsubseteq e'_{i'}$.* □

This notion of containment has also been used previously for Verso relations [5], partial information [8], and or-sets [29]. It is a particular case of the lower (Hoare) powerdomain ordering [18], and it coincides with the simulation relation between complex objects represented as graphs [6, 7]. It is also the smallest order relation for XML instances which is reflexive (*i.e.* $e \sqsubseteq e$), is a congruence (*i.e.* $e_1 \sqsubseteq e'_1 \wedge \dots \wedge e_n \sqsubseteq e'_n$ implies $[t, \{e_1, \dots, e_n\}] \sqsubseteq [t, \{e'_1, \dots, e'_n\}]$, $\{e_1\} \sqsubseteq \{e'_1\}$, and $e_1 \cup e_2 \sqsubseteq e'_1 \cup e'_2$), and satisfies the empty set $\emptyset \sqsubseteq e$.

The containment of two XML instances can be justified by tree homomorphism (not necessarily injective). Following [30], we define an *embedding* from a tree t_1 to t_2 as a node mapping, which is root preserving, respects node relationships and edge labels. It is trivial to verify that two XML instances e and e' satisfy $e \sqsubseteq e'$, if and only if e's corresponding tree can be embedded in e''s corresponding tree.

XML instance containment is reflexive and transitive, but not antisymmetric: $e \sqsubseteq e'$ and $e' \sqsubseteq e$ do not imply $e = e'$. As an example, consider the three XML instances in Figure 3. They contain each other pairwise, but are not equivalent.

In the development of our algorithms we will use the notion of a *minimal* XML object, defined as follows.

Definition 2.4 (Minimal XML Object). *Let e be an XML object. e is said to be minimal if there is no XML object $[t, \{e_1, \dots, e_n\}]$ nested in e, where there exist $i, j \in [1, n], i \neq j$, and $e_i \sqsubseteq e_j$.* □

Proposition 2.5. *Containment on minimal XML objects is a partial order.*

Proof: As for general XML objects, containment on minimal XML objects is reflexive and transitive. Below we prove it is also anti-symmetric; i.e., given two minimal XML objects, D and D', $D = D'$ if and only if $D \sqsubseteq D'$ and $D' \sqsubseteq D$.

Assume to the contrary, $D \neq D'$. $D \sqsubseteq D'$ and $D' \sqsubseteq D$ imply the existence of a tree embedding from D to D', denoted as e_1, and a tree embedding from D' to D, denoted as e_2. Because $D \neq D'$, there must exist a node which is *not* mapped to itself through e_1 and e_2. Suppose n_1 of D is such a pair, and all its parents are mapped to themselves through e_1 and e_2. Suppose e_1 maps n_1 to n_2, but e_2 maps n_2 to another node $n'_1 \neq n_1$. According to the assumption, n_2's parent is mapped to n_1's parent, so n'_1 must be one of n_1's sibling node. Accordingly, the subtree rooted at n_1 is embedded in the subtree rooted at n_2 and transitively embedded in the subtree rooted at n'_1. Therefore, D is not minimal, which contradicts our assumption. □

Intuitively, an XML object that is not minimal contains redundant information. We can *minimize* it by removing redundancy. We define a weaker equivalence relationship between XML objects by comparing their minimized forms. As stated by Proposition 2.9 this notation of equivalence agrees with bi-directional XML object containment.

Definition 2.6 (Minimized Form of an XML object). *Let e be an XML object and e' be a minimal XML object. e' is called a* minimized form *of e, if $e \sqsubseteq e'$ and $e' \sqsubseteq e$.*

Proposition 2.7. *Every XML object has one and only one minimized form.*

Proof: We first show the existence by *minimizing* D to a minimal XML object. We process D bottom-up. For each two sibling subtrees T and T' in D, where $T \sqsubseteq T'$, we remove T from D. The process halts when it reaches the root. It's easy to verify that the result, D', is a minimal XML object.

Next, we show the uniqueness. Assume to the contrary, D can be minimized into two different minimal XML objects D_1 and D_2. $D_1 \sqsubseteq D$ and $D \sqsubseteq D_2$ imply $D_1 \sqsubseteq D_2$; similarly $D_2 \sqsubseteq D_1$. Based on Proposition 2.5, $D_1 = D_2$, which contradicts our assumption.

Definition 2.8 (XML Object Weak Equivalence). *Let e and e' be two XML objects. e is* weak equivalent *to e', denoted as $e \doteq e'$, if the minimized form of e is equivalent to the minimized form of e'.*

Proposition 2.9. *Let e and e' be two XML instances. $e \doteq e'$, iff $e \sqsubseteq e'$ and $e' \sqsubseteq e$.*

Proof: Let M be the minimized form of D, and M' be the minimized form of D'.

if: $D \sqsubseteq D'$ implies $M \sqsubseteq D \sqsubseteq D' \sqsubseteq M'$, so $M \sqsubseteq M'$. Similarly, $D' \sqsubseteq D$ implies $M' \sqsubseteq M$. Because M and M' are minimal, from above $M = M'$. So $D \doteq D'$.

only if $D \doteq D'$ implies $M = M'$. $D \sqsubseteq M = M' \sqsubseteq D'$, so $D \sqsubseteq D'$; similarly, $D' \sqsubseteq D$.

Based on the definition of XML object containment and weak equivalence, we define containment and weak equivalence of c-XQueries as follows.

Definition 2.10 (c-XQuery Containment). *Let Q and Q' be two c-XQueries. Q is* contained in *Q', denoted as $Q \sqsubseteq Q'$, if for every input XML instance D, $Q(D) \sqsubseteq Q'(D)$.* □

Proposition 2.11. *Let Q and Q' be two c-XQueries. $Q \doteq Q'$ iff $Q \sqsubseteq Q'$ and $Q' \sqsubseteq Q$.*

3 Containment of cc-XQueries

We begin by considering query containment for cc-XQueries. A cc-XQuery does not contain tag variables, thus can be viewed as a generalization of a boolean conjunctive query (i.e, a conjunctive query with an empty head), except that they 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 number of canonical answers we need to consider, which later carry over to c-XQueries.

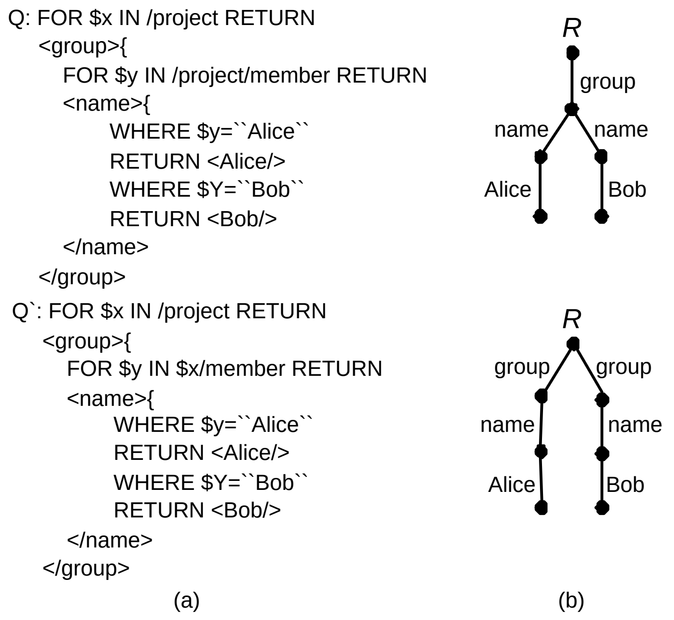
Intuitively, given a pair of queries Q and Q', we cannot check that for *every* possible XML input D, $Q(D) \sqsubseteq Q'(D)$. Hence, our goal is to find a finite set of representative inputs, called *canonical databases*, which have the property that $Q \sqsubseteq Q'$ if and only if $Q(DB) \sqsubseteq Q'(DB)$ 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. Initially, one could conjecture that it suffices to consider all the answers corresponding to subtrees of the head tree of Q that contain the root. However, the following example refutes this conjecture. Furthermore, it shows that the result in [28] offers only a necessary condition for query containment, but not a sufficient one.

Example 3.1: Consider the two cc-XQueries, Q and Q', in Figure 4(a) (and in our syntax below).

```
Q:  g ← $\Re$p$X'$ {                    Q':  g ← $\Re$p$X$ {
        n ← $\Re$p$X$, $X$m$Y$ {                 n ← $X$m$Y$ {
            a ← $Y$a$Z$                              a ← $Y$a$Z$
            b ← $Y$b$W$                              b ← $Y$b$W$
    }}                                       }}
```

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 or not they are working on the same project. Figure 4(b) shows the results of Q and Q' on the XML instance D of Example 2.2. $Q(D) \not\sqsubseteq Q'(D)$, and thus $Q \not\sqsubseteq Q'$. In contrast, containment *does* hold for canonical databases generated for all subtrees of the head tree. □



```
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>
```

Figure 4: Example 3.1: (a) Q and Q'; (b) the answers to Q and Q'.

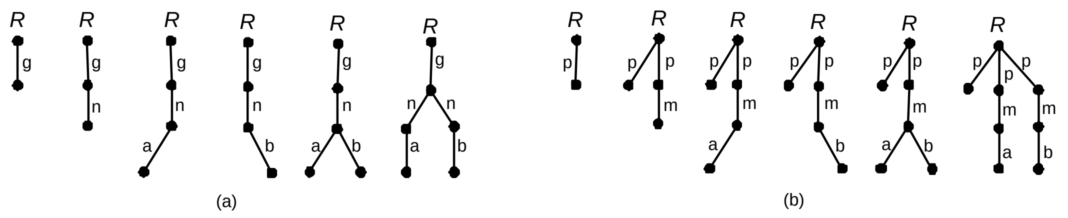


Figure 5: Example 3.1: (a) Q's canonical answers; (b) Q's canonical databases.

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 subtrees of the head tree).

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 $CA \sqsubseteq H$.* $\square$

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 X be a fresh node variable in $\hat{q}_n$. There is a distinct node in DB_{CA} for X, denoted as N_X.*
- *Let XlY be a conjunct in $\hat{q}_n$. There is an edge labeled l from N_X to N_Y.* $\square$

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 5(a) shows six canonical answers for Q in Example 3.1. Figure 5(b) shows the corresponding canonical databases.

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.

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.

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 $CA \sqsubseteq Q(D)$.* □

Proof: Each $Q(D)$ has a unique minimized form, denoted as A. Let H be the head tree of Q. $Q(D) \sqsubseteq H$ and $A \sqsubseteq Q(D)$ imply that $A \sqsubseteq H$. So A is a canonical answer of Q. □

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 $DB_{CA} \sqsubseteq D$.* □

Proof: *if:* $DB_{CA} \sqsubseteq D$ implies that there exists a tree homomorphism $\sigma : DB_{CA} \to D$.

Let n be a node in CA generated by query block $\hat{q_n}$ in Q, and $\bar{N}$ be the nodes in DB_{CA} associated with n and n's ancestors. There exists a substitution $\varphi_{DB} : \hat{q_n} \to \bar{N}$. Composing it with σ, results in a substitution $\varphi = \varphi_{DB} \circ \sigma : \hat{q_n} \to \sigma\bar{N}$. The result node obtained from the substitution is the node to which n maps. This proves $CA \sqsubseteq Q(D)$.

only if: $CA \sqsubseteq Q(D)$ implies that there exists a tree homomorphism $\sigma : DB_{CA} \to D$. Let n be a node in CA. $\sigma(n)$ is n's corresponding node in $Q(D)$. The definition of a $cc - XQuery$ guarantees that n and $\sigma(n)$ are generated by the same query block, denoted as $hatq_n$. The definition of a canonical database DB indicates a one-to-one correspondence between the nodes associated with n and the node variables in $\hat{q_n}$, denoted as $\phi : DB_{CA} \to \hat{q_n}$. Besides, there exists a substitution $\varphi : \hat{q_n} \to D$. Compose ϕ with φ results in the tree homomorphism from DB_{CA} to D. Thus, $DB_{CA} \sqsubseteq D$.

Proof for Theorem 3.4:
(1) ⇒ (2): Follows from the definition.

(2) ⇒ (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) ⇒ (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, $Q(D) \sqsubseteq Q'(D)$. □

The third condition of Theorem 3.4 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 (which is a sufficient and necessary condition for containment) 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 fan-out of a query block in Q, and let d be the nesting depth of Q. We next 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). Because the tag of a node's incoming edge has m options, the number of canonical answers is similar to the above complexity, but with d exponents. We obtain this by considering the upper and lower bounds of the maximal size of canonical answers.

Consider a query block $\hat{q}$ with depth $d-1$ in Q, and its parent query block $\hat{p}$. The query block $\hat{q}$ has no more than m children query blocks, each of which is a leaf block. Thus, the subtrees rooted at the nodes generated by $\hat{q}$ have no more than 2^m different shapes; and the node generated by $\hat{p}$ has no more than 2^m children that are generated by $\hat{q}$. Since $\hat{p}$ has up to m children query blocks, it can have up to $m \cdot 2^m$ children subtrees. For a similar reason, there are no more than $2^{(m \cdot 2^m)}$ different shapes for subtrees rooted at the nodes generated by $\hat{p}$; thus, $\hat{p}$'s parent can have up to $m \cdot 2^{(m \cdot 2^m)}$ children. Iteratively, the top-level query block can have no more than $m \cdot 2^{m \cdot \left(2^{\cdot^{\cdot^{m \cdot (2^m)}}}\right)}$ children, where the number of the exponents is $d-1$. This is the upper bound of the maximal size of canonical answers.

The real size of canonical answers is far less than the upper bound because of the concern for sibling tree containment; however, the maximal size has a lower bounded in the same class as the upper bound. Again, we analyze bottom-up. Consider a node generated by $\hat{q}$. Among the trees rooted by the node, $\binom{m}{\lfloor m/2 \rfloor} \geq 2^{\lfloor m/2 \rfloor}$ have exactly $\lfloor m/2 \rfloor$ leaf nodes. These subtrees do not contain each other pairwise. A node generated by $\hat{p}$ can have any of them as subtrees. The query block $\hat{p}$ has up to m children query blocks, so among the subtrees rooted at a node generated by $\hat{p}$, at least $2^{(m \cdot \binom{m}{\lfloor m/2 \rfloor})} \geq 2^{(m \cdot (2^{\lfloor m/2 \rfloor}))}$ do not contain each other. Iteratively, among the subtrees rooted at a node generated by a query with depth i, there are at least $t_i = 2^{(m \cdot \binom{t_{i+1}}{\lfloor t_{i+1}/2 \rfloor})} \geq 2^{(m \cdot (2^{\lfloor t_{i+1}/2 \rfloor}))}$ shapes. When the depth is 2, t_2 is in class $O(2^{m \cdot \left(2^{\cdot^{\cdot^{m \cdot (2^{(m/2)})}}}\right)})$, which is a tower of powers with height $d-1$. This implies that the lower bound is $O(m \cdot 2^{m \cdot \left(2^{\cdot^{\cdot^{m \cdot (2^{(m/2)})}}}\right)})$. As a special case, when $m = 1$, the maximum size of a canonical answer equals the depth of the query. This agrees with the discussion in the next section.

3.3 Effect of the Fanout

In this section, we show that fanout 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 (PTIME). *Let Q and Q' be cc-XQueries. If either of them is linear, testing $Q \sqsubseteq Q'$ is in PTIME.* □

Proof: 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. Consequently, the size of a canonical database for Q is bounded by $|Q|$, and that for Q' is bounded by $|Q'|$.

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. Specifically, the time complexity for one canonical database embedding checking is $O(|Q| \cdot |Q'|)$, and that for containment checking is $O(d \cdot |Q| \cdot |Q'|)$. □

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.

The following theorem shows that the restriction on the fanout is critical.

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

Proof: We reduce the 3CNF problem to query containment of cc-XQueries with depth 2.

Let ψ be a 3-CNF formula with variables $x_1, \ldots, x_n$ and clauses $c_1, \ldots, c_m$. We construct two cc-XQueries, Q and Q', s.t. ψ is satisfiable iff $Q \not\sqsubseteq Q'$. We show that ψ is satisfied by an assignment ν, iff we can construct a canonical answer CA of Q from ν, which is also a canonical answer of Q', but violates $DB'_{CA} \sqsubseteq DB_{CA}$. Hence, the proof follows from the third condition of Theorem 3.4.

Both Q and Q' have two levels: a top-level query block, and $2n$ children blocks. In each query, the top level block returns the tag u; for each variable $x_i, i \in [1,n]$, there are two children query blocks on the second level, returning v_i and w_i respectively. Hence, Q and Q' have exactly the same canonical answers.

In query Q, there are two groups of conjuncts. Group A contains m^2 conjuncts in the top level query block. For each clause $c_i, i \in [1,m]$, there are m conjuncts: $\Re a C_i$, and $C_i d_j\_$ for each $j \in [1,m], j \neq i$. (_ indicates a node variable) In group B, there are n conjuncts in the top level query block. For each variable $x_i, i \in [1,n]$, there is a conjunct $\Re a X_i$. On the second level, for each variable x_i, there is a conjunct $X_i d_1\_$ in the query block returning v_i, and $m-1$ conjuncts $X_i d_j\_, j \in [2,m]$, in the query block returning w_i.

In query Q', the top-level query block has the conjunct $\Re a X$. On the second level, if x_i occurs in clause c_j, there is a conjunct $X d_j\_$ in the query block that returns v_i; if $\bar{x}_i$ occurs in clause c_j, there is a conjunct $X d_j\_$ in the query block that returns w_i.

As an example, consider the formula $\psi = (x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$. We build the following two queries:

```
Q:  for $\Re aC_1, C_1d_2\_, C_1d_3\_, \Re aC_2, C_2d_1\_, C_2d_3\_, \Re aC_3, C_3d_1\_, C_3d_2\_, \Re aX_1, \Re aX_2, \Re aX_3, \Re aX_4$ return
    [u, {
        for $X_1d_1\_$ return
        [$v_1$,{}]
        for $X_1d_2\_, X_1d_3\_$ return
        [$w_1$,{}]
        for $X_2d_1\_$ return
        [$v_2$,{}]
        for $X_2d_2\_, X_2d_3\_$ return
        [$w_2$,{}]
        for $X_3d_1\_$ return
        [$v_3$,{}]
        for $X_3d_2\_, X_3d_3\_$ return
        [$w_3$,{}]
        for $X_4d_4\_$ return
        [$v_4$,{}]
        for $X_4d_2\_, X_4d_3\_$ return
        [$w_4$,{}]
    }]
Q': for $\Re aX$ return
    [u, {
        for $Xd_1\_, Xd_3\_$ return
        [$v_1$,{}]
        for $Xd_2\_$ return
        [$w_1$,{}]
        for $Xd_1\_, Xd_2\_, Xd_3\_$ return
        [$v_2$,{}]
        for $true$ return
        [$w_2$,{}]
        for $Xd_2\_, Xd_3\_$ return
        [$v_3$,{}]
        for $true$ return
        [$w_3$,{}]
        for $true$ return
        [$v_4$,{}]
        for $Xd_1\_$ return
```

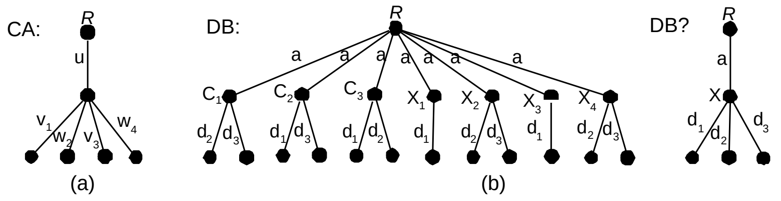


Figure 6: (a) An example canonical answer CA of Q that corresponds to an assignment ν, in which $x_1 = true, x_2 = false, x_3 = true, x_4 = false$; (b) Q and Q''s canonical databases for CA.

```
        [w4,{}]
    }]
```

We construct CA from ν, as follows. Consider a two-level canonical answer, which has a single node with incoming edge labeled u on the first level. There is a node with incoming edge labeled v_i, iff $\nu(x_i) = true$; there is a node with incoming edge labeled w_i, iff $\nu(x_i) = false$. (Note that if for a given $i \in [1, n]$, neither v_i nor w_i occurs in CA, then in the corresponding assignment, x_i can be assigned either *true* or *false*.) Based on this, we partition these canonical answers into three classes. If both v_i and w_i occur in CA, we say the canonical answer corresponds to an *invalid* assignment. Otherwise, the canonical answer must correspond to a valid assignment. If the assignment does not satisfy ψ, we say the canonical answer corresponds to a *bad* assignment. If the assignment satisfies ψ, we say the canonical answer corresponds to a *good* assignment. In our example, a canonical answer that corresponds to the assignment $x_1 = true, x_2 = false, x_3 = true, x_4 = false$ is shown in Figure 8(a).

Constructing Q and Q' takes polynomial time. Now we show that $Q \not\sqsubseteq Q'$ iff ψ is satisfiable.

if Assume ν is an assignment that satisfies ψ. Consider the canonical answer corresponding to ν, which is a good assignment. The canonical answer in Figure 8(a) corresponds to a good assignment for the given example. The two canonical databases, DB_{CA} and DB'_{CA}, are shown in Figure 8(b).

Since ν satisfies ψ, each clause $c_j = true, j \in [1, m]$. So at least one of its three literals is satisfied by ν. Accordingly, the edge corresponding to the literal occurs in CA, and in turn d_j must occur in DB'_{CA}. Thus, in DB'_{CA} there exists a *full-width node* X with depth 1, which has incoming edge labeled a, and outgoing edges labeled $d_1, \ldots, d_m$ respectively.

On the other hand, DB_{CA} does not contain any full-width nodes. Thus, $DB'_{CA} \not\sqsubseteq DB_{CA}$, and so $Q \not\sqsubseteq Q'$.

only if Assume ψ is not satisfiable. Then a two-level canonical answer CA of Q must either correspond to an invalid assignment or correspond to a bad assignment.

If CA corresponds to an invalid assignment, then there exists $i \in [1, n]$, where both v_i and w_i occur in CA. The Group-B conjuncts of Q guarantee that there will be a full-width node in DB_{CA}. Thus DB'_{CA} must be contained in DB_{CA}.

If CA corresponds to a bad assignment, then in DB'_{CA} there do not exist full-width nodes. On the other hand, the Group A conjuncts of Q ensures $DB'_{CA} \sqsubseteq DB_{CA}$ as far as DB'_{CA} does not contain full-width nodes. So again, $DB'_{CA} \sqsubseteq DB_{CA}$.

Finally, it's easy to verify that for the empty canonical answer CA_0 and the one-level canonical answer CA_1 of Q, we have $DB'_{CA_0} \sqsubseteq DB_{CA_0}, DB'_{CA_1} \sqsubseteq DB_{CA_1}$. This proves $Q \sqsubseteq Q'$. □

3.4 cc-XQueries with Fixed Nesting Depth

In this section we show that for *any* fixed nesting depth, query containment for cc-XQueries 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.* □

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 conjuncts 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. Then p appears in at most $cd - 1$ siblings of N.* $\square$

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: In the proof, we call a canonical answer CA a *violator* if it violates the condition $DB'_{CA} \sqsubseteq DB_{CA}$. We call a violator CA a *minimum violator*, if CA does not have any proper subtree $\widetilde{CA}$, which also violates $DB'_{\widetilde{CA}} \sqsubseteq DB_{\widetilde{CA}}$. We prove the theorem by showing that a non-kernel canonical answer cannot be a minimum violator. By induction, if CA is a non-kernel canonical answer that violates $DB'_{CA} \sqsubseteq DB_{CA}$, there must exist a kernel canonical answer KCA, which is a proper subtree of CA, and violates $DB'_{KCA} \sqsubseteq DB_{KCA}$. So even without checking containment on CA, we can still get the right conclusion that $Q \not\sqsubseteq Q'$.

Assume to the contrary, there exists a non-kernel canonical answer CA, which is a minimum violator for Q and Q'. Let N be the node in CA with $l > cd$ children, $C_1, \ldots, C_l$. We denote the subtrees rooted at $C_1, \ldots, C_l$ as $T_1, \ldots, T_l$. Each of them contains a common path pattern p.

Consider the nodes in DB'_{CA} that correspond to N or N's ancestors, i.e. correspond to the node variables in the query block of Q' that generate N or N's ancestors. According to the structure of CA, in DB'_{CA} there must be nodes that have no less than l children, where each subtree $T_i, i \in [1, l]$, generates at least one of the children together with the subtree rooted at that child. We call those nodes *l-children nodes*. Below we discuss l-children nodes, their ancestors, their descendants, and the rest nodes in DB'_{CA} separately, showing that nodes in each category can be mapped to some nodes in DB_{CA}. This implies $DB'_{CA} \sqsubseteq DB_{CA}$.

- We first consider an l-children node in DB'_{CA}. If we remove a subtree $T_i, i \in [1, l]$ from CA, and accordingly Y's descendants in DB'_{CA} corresponding to T_i, the result subtree rooted at Y must be embedded in a subtree in DB_{CA}, because CA is a minimal violator. We denote the root of the subtree in DB_{CA} as Y_i. If there exist $Y_i, Y_j, i \neq j$ that are the same node, Y can be mapped to it. Below we show that those Y_i's, $i \in [1, l]$, can not be all different; otherwise, the number of nodes in DB_{CA} exceeds the maximum number of nodes in Q's canonical database for CA.

  Suppose there are k leaves in CA. Each leaf corresponds to at least one descendant node of Y; otherwise, removing that leaf will not affect the subtree rooted at Y, and Y still cannot be mapped to any node in DB_{CA}, which implies CA is not a minimal violator. As a result, Y has at least k descendant nodes. Each descendant node that corresponds to a node in T_i, can be mapped to a descendant of $Y_j, j \in [1, l], j \neq i$. If $Y_1, \ldots, Y_l$ are different, their descendants are disjoint. So each descendant node has $l - 1$ counterparts in DB_{CA}. Plus $Y_1, \ldots, Y_l$, we have at least $l + k(l - 1)$ nodes in DB_{CA}.

  On the other hand, there are at most kd nodes in CA. Each node has a generator query block; each query block contains no more than c conjuncts; and each conjunct introduces no more than one new node variable, and in turn no more than one node to the canonical database. So there can be no more than $kdc \leq k(l - 1) < l + k(l - 1)$ nodes in DB_{CA}, which leads to contradiction.

- In the same fashion we prove an ancestor of an l-children node must be able to map to a node in DB_{CA}. We denote the ancestor as X and its l-children child as Y. Similarly, since CA is a minimal violator, after removing $T_i, i \in [1, l]$, and accordingly Y's descendants corresponding to T_i, X must be able to map to a node in DB_{CA}, denoted as X_i. There must exist two $X_i, X_j, i \neq j$ that are the same node; otherwise, for the same reason as discussed for l-children nodes, there are at least $l + k(l - 1)$ nodes in DB_{CA}; but no more than $kdc < l + k(l - 1)$ of them can correspond to nodes in CA, which leads to contradiction. Therefore, X can be mapped to that X_i.

- If a node, denoted as X, is a descendant of an l-children node, it corresponds to a node in one $T_i, i \in [1, l]$. Removing other T_i's from CA does not affect the subtree rooted at X in DB'_{CA}. Given CA is a minimal violator, X must be able to map to a node in DB_{CA}.
- Finally, consider the rest of the nodes, denoted as X. In the subtree rooted at X, any node cannot correspond to the node at the very end of the path pattern p in CA. Otherwise, X must be an l-children node, or an ancestor, or a descendant of an l-children node. Thus, removing the nodes at the very end of the path pattern p in CA does not affect the subtree in DB'_{CA} rooted at X. Given CA is a minimal violator, X must be able to map to a node in DB_{CA}.

From the above analysis, all nodes in DB'_{CA} can be mapped to some node in DB_{CA}. This contradicts with the assumption that $DB_{CA} \not\sqsubseteq DB_{CA}$. □

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.* □

Proof: 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. □

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 PTIME, 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 $\bar{I}_m$.

Formally, simulation is defined as follows.

Definition 4.1 (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:*

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

In [28] it is shown that query simulation can be checked by establishing a *simulation mapping* from Q' to Q, defined as follows. We denote with $\mathcal{I}_j$ the set of index variables in $\bar{I}_j, j \in [1, m]$, with $\mathcal{I}$ the set of all index variables, with $\mathcal{V}$ the set of non-index variables (head and non-head) of Q, and with $\mathcal{C}$ the set of constants. For each $j = 1, \ldots, m$, the *witness variables at level* j are a fresh copy of $\mathcal{I}_{j+1} \cup \cdots \cup \mathcal{I}_m \cup \mathcal{V}$, denoted $\mathcal{W}_j$; in particular, $\mathcal{W}_m$ is a fresh copy of $\mathcal{V}$. We denote Q^w_j a copy of Q in which the variables in $\mathcal{I}_{j+1} \cup \cdots \cup \mathcal{I}_m \cup \mathcal{V}$ have been replaced with those in $\mathcal{W}_j$, and let $Q^w = Q^w_1 \cup \cdots \cup Q^w_m$. Then a *simulation mapping from Q' to Q of depth m* is a mapping $\varphi : \mathcal{I}' \cup \mathcal{V}' \to \mathcal{I} \cup \mathcal{W}_1 \cup \cdots \cup \mathcal{W}_m \cup \mathcal{V} \cup \mathcal{C}$ such that (1) $\varphi(Body(Q')) \subseteq Body(Q) \cup Body(Q^W)$, (2) $\varphi(\bar{V}') = \bar{V}$, and (3) $\varphi(\mathcal{I}'_j) \cap (\mathcal{I}_{j+1} \cup \cdots \cup \mathcal{I}_m \cup \mathcal{W}_{j+1} \cup \mathcal{W}_m \cup \mathcal{V}) = \emptyset, j \in [1, m]$. 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 for the indexed conjunctive queries we will create is in PTIME.

Compare a simulation mapping to a classical containment query mapping τ, where each variable in Q' can be mapped to any variable in Q, s.t. (1) $\tau(Body(Q')) \subseteq Body(Q)$, (2) $\tau(\bar{V}') = \bar{V}$. As shown in the following prepositions, query simulation is a generalization of query containment for indexed conjunctive queries. Simulation reduces to query containment when there are no index variables.

Proposition 4.2. *Let Q and Q' be two flattened queries. If there is a simulation mapping from Q' to Q, there is a query mapping from Q' to Q.* $\square$

Proof: Given a simulation mapping φ from Q' to Q, we construct a classical containment mapping τ. Construct mapping $\psi : \mathcal{I} \cup \mathcal{W}_1 \cup \cdots \cup \mathcal{W}_m \cup \mathcal{V} \cup \mathcal{C} \to \mathcal{I} \cup \mathcal{V} \cup \mathcal{C}$, sending each witness variable to its original variable in the index, each witness conjunct in Q^w to its original conjunct in the body of Q, and other variables and conjuncts to themselves. Compose φ with ψ results in $\tau = \varphi \circ \psi : \mathcal{I}' \cup \mathcal{V}' \to \mathcal{I} \cup \mathcal{V} \cup \mathcal{C}$. It's easy to verify τ is a mapping from Q' to Q. $\square$

Proposition 4.3. *Let Q and Q' be two flattened queries where $\bar{V} = \emptyset$. There is a simulation mapping from Q' to Q, iff $\bar{V}' = \emptyset$, and there exists a query mapping from Q' to Q.* $\square$

Proof: The *only if* direction holds according to Proposition 4.2. We prove the *if* direction. Given a classical query mapping τ from Q' to Q, we construct a simulation mapping φ. Let $\psi : \mathcal{I} \to \mathcal{I}_1 \cup \mathcal{W}_1$ be a mapping where each index variable in $\mathcal{I}_1$ is mapped to itself, and other index variables are mapped to their witnesses in $\mathcal{W}_1$. Compose τ with ψ results in $\varphi = \tau \circ \psi : \mathcal{I}' \cup \mathcal{V}' \to \mathcal{I} \cup \mathcal{W}_1 \cup \mathcal{V} \cup \mathcal{C}$. It's easy to verify φ is a simulation mapping from Q' to Q. $\square$

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 4.4 (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*

$$IQ_{CA}(\bar{I}_1; \ldots; \bar{I}_{d-1}; V) : -X_1R_1Y_1, \ldots, X_nR_nY_n,$$

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 $\hat{q}_n$.*
- *$\bar{V}$ includes the returned tag variable in $\hat{q}_n$, if any.*
- *The body includes every conjunct in $\hat{q}_n$.* □

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

Proof: From the definition of query simulation, it is easy to verify that conditions (1)(2) hold, iff for every canonical answer CA and its canonical database DB_{CA}, we have $Q(DB_{CA}) \sqsubseteq Q'(DB_{CA})$. Composing this with Theorem 3.4 gives that conditions (1)(2) hold iff $Q \sqsubseteq Q'$. □

Example 4.6: Consider query Q in the example of section 2. Given the head tree, shown in Figure 2(b), as the canonical answer CA, the indexed conjunctive query for CA is the following (note that 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 5(a), denoted as CA', is also a canonical answer of Q by applying variable isomorphism. The indexed conjunctive query for CA' is the following:

$$IQ_{CA'}(X; Y_1, Y_2; \emptyset) : -\Re pX, XmY_1, XmY_2, Y_1aZ_1, Y_2bZ_2$$

By applying Theorem 4.5, we can justify that the query Q' in Example 3.1 is contained in the above query Q, but not the other way around. □

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 is slightly weaker than Theorem 3.7. We require that Q has fanout 1, rather than this holding for *either Q or Q'*.)

Theorem 4.7. *Let Q and Q' be c-XQueries.*

- *If Q has a maximal fanout of 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.* □

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.

5.1 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. We call such queries *u-XQueries*.

We start with constant queries, which may contain unions only in the first form. Each of such queries, Q, can be rewritten as a union of cc-XQueries as follows:

1. Suppose Q contains a subquery in the shape of

$$Q' ::= \textbf{for } B \textbf{ return } [\mathcal{T}, \{Q_1, Q_2, \ldots, Q_n\}],$$

   where there exist $i, j, i, j \in [1, n], i \neq j$, s.t. Q_i and Q_j return the same tag constant. Then the union of cc-XQueries for Q is the union of all cc-XQueries for Q where Q' does not contain Q_i, and all cc-XQueries for Q where Q' does not contain Q_j.

2. If Q is a cc-XQuery, then the union of Q is Q itself.

Similar to Theorem 3.4, we prove the following theory.

Theorem 5.1. *Let $Q = \bigcup_{i=1}^{n} CQ_i$ and $Q' = \bigcup_{j=1}^{m} CQ'_j$ be two u-XQueries composed of unions of cc-XQueries. The following three conditions are equivalent:*

1. *$Q \sqsubseteq Q'$;*
2. *for every $i \in [1, n]$ and every canonical database DB of CQ_i, $Q(DB) \sqsubseteq Q'(DB)$;*
3. *for every $i \in [1, n]$ and every canonical answer CA of CQ_i, there exists $j \in [1, m]$, s.t. (1) CA is a canonical answer of CQ'_j; and (2) $DB'_{j,CA} \sqsubseteq DB_{i,CA}$.* □

In a similar fashion, we consider the second form of union. To simplify, we first assume in the query there is no union in the first form. We rewrite each query, Q, as a union of c-XQueries as follows:

1. Suppose Q is a query in the shape of

$$Q ::= \textbf{for } B \textbf{ return } [\mathcal{T}, \{Q_1, Q_2, \ldots, Q_n\}],$$

   where there exists $i, i \in [1, n]$, s.t. Q_i returns a tag variable. Then the union of cc-XQueries for Q is the union of all c-XQueries for Q where Q' only contains Q_i, and all c-XQueries for Q where Q' does not contain Q_i.

2. If Q is a cc-XQuery, then the union of Q is Q itself.

For a general u-XQuery, we combine the above to rewrite it as a union of c-XQueries. Note that the number of the c-XQueries is linear in terms of the size of the u-XQuery. Similarly, we have the following theorem.

Theorem 5.2. *Let $Q = \bigcup_{i=1}^{n} CQ_i$ and $Q' = \bigcup_{j=1}^{m} CQ'_j$ be two u-XQueries composed of unions of c-XQueries. Containment $Q \sqsubseteq Q'$ holds iff for every $i \in [1, n]$, and every canonical answer CA of CQ_i, there exists $j \in [1, m]$, s.t. (1) CA is also a canonical answer of CQ'_j, and (2) $IQ_{i,CA} \preceq IQ'_{j,CA}$.*

Next, we show that adding unions to c-XQueries does not increase the complexity of the containment problem. The key ingredient is to show that checking simulation on an existence quantifier of Theorem 5.1 can still be done in polynomial time in the size of the canonical database.

Theorem 5.3. *Let Q and Q' be u-XQueries. Query containment is in PTIME when Q has nesting depth of 1, or Q has maximal fanout of 1; query containment is coNP-complete if Q has a fixed nesting depth.* □

Proof: The lower bounds are no less than that of the query containment problem for c-XQueries. Now we prove the upper bound.

In case (1) and (2), the number and size of canonical databases are bounded by the size of Q and Q'.

Given CA, a canonical answer of Q, we construct the indexed queries for CA, denoted as IQ_{CA} and IQ'_{CA}, in the same way as stated in Definition 4.4, except that when constructing IQ'_{CA}, (1) for each node of CA, we repeat the process for every candidate generator query block; (2) let $\hat{q}_1$ and $\hat{q}_2$ be two disjunctive query blocks; we mark variables for $\hat{q}_1$ as a *back-up* of variables for $\hat{q}_2$. Now we find the simulation from IQ'_{CA} to IQ_{CA}. This can be reduced to a tree embedding problem, where the embedding condition is loosened so that a node P can be mapped to a node Q as far as P's children or the respective back-up children can be mapped to children of Q. Establishing such an embedding is in polynomial time of the size of the canonical database, thus in polynomial time of the query size.

In case (3), let $Q = \bigcup_{i=1}^{n} CQ_i$ and $Q' = \bigcup_{j=1}^{m} CQ'_j$ be two u-XQueries. We describe a non-deterministic polynomial algorithm, that checks whether there exists $i \in [1, n]$ and a canonical answer of CQ_i, none of $j \in [1, m]$ satisfies both conditions (1)(2) in Theorem 5.1. We guess $i \in [1, n]$ and a canonical answer CA. Similar to the cases of (1) and (2), it takes polynomial time to decide whether none of $j \in [1, m]$ satisfies both conditions (1)(2). So the problem is in co-NP. □

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 have the following result.

Theorem 5.4. *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 4.7 still apply. □

The difference between Theorem 5.3 and Theorem 5.4 is that the lower bound of case (1) and (2) rises to co-NP hard. 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.

5.2 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*$^\neg$.

Following the spirit of containment checking for relational queries[27], every canonical answer of a c-XQueries$^{\neg n}$ may correspond to several indexed queries, and we need to check containment on each of them. We construct the indexed queries for a canonical answer CA of Q as follows: (1) Construct a *model indexed query* as in Definition 4.4, ignoring the negated conjuncts. (2) Consider all possible partitions of node variables in the model indexed query. For each one, we merge all nodes in the same group into a single node (without destroying the tree structure), resulting in an indexed query. For query containment $Q \sqsubseteq Q'$ to hold, it must be the case that every indexed conjunctive query IQ_{CA} of Q either violates the negation predicates in Q, or satisfies $IQ_{CA} \preceq IQ'_{CA}$.

Example 5.5: Consider the following c-XQueries$^\neg$ $Q^\neg$ with negation $\neg YbW$.

```
$Q^\neg$: for $\Re pX'$ return
    [g, {
        for $\Re pX, XmY$ return
        [n, {
```

```
    |g            p/|p\p          p/\p       p/|p          |p
  n/ \n          X' X1 X2      (X')X1 X2    X' X1(X2)      X1(X2,X')
  a|  |b            |m |m          |m |m         |m          |m
                    Y1 Y2          Y1 Y2       Y1(Y2)      Y1(Y2)
                    |a |b          |a |b      a/ \b       a/ \b
                    Z  W           Z  W       Z   W       Z   W
   (a)            (b)                          (c)
```

Figure 7: (a)$Q^{\neg}$'s canonical answers, CA; (b)the model canonical database for CA; (c) extended canonical databases for CA.

```
            for $YaZ, \neg YbW$ return
            [a, {}]
            for $YbW$ return
            [b, {}]
        }]
    }]
```

Since $Q^{\neg}$ does not return tag variables, we discuss canonical database to simplify. Given the canonical answer in Figure 7(a), the model canonical database obtained according to the definition for c-XQueries is shown in Figure 7(b). Without destroying the tree structure, we can group nodes among X', X_1 and X_2; if we group nodes X_1 and X_2, we also need to group nodes Y_1 and Y_2. The result canonical databases for this particular canonical answer are shown in Figure 7(b)(c).

We compare $Q^{\neg}$ with Q in Example 3.1. $Q^{\neg}$ is the same as Q except that it has an additional negative predicate.

We first check whether $Q^{\neg} \sqsubseteq Q$. Consider the canonical answer CA shown in Figure 7(a), and its corresponding canonical databases shown in Figure 7(b)(c). The last two canonical databases in (c) do not satisfy the negation predicates in Q. The rest one in (c) and the one in (b) both satisfy $DB'_{CA} \sqsubseteq DB_{CA}$. In a similar fashion we check canonical databases for other canonical answers of Q, and conclude $Q^{\neg} \sqsubseteq Q$.

Now we check whether $Q \sqsubseteq Q^{\neg}$. CA in Figure 7(a) is also a canonical answer of Q', and its corresponding canonical databases of Q' are the same as those of Q, shown in Figure 7(b)(c). Because Q' does not contain any negation predicates, we need to check $DB'_{CA} \sqsubseteq DB_{CA}$ on all of the four canonical databases. Because of the negation predicates in Q, the containment does not hold for the last two canonical databases in Figure 7c. This implies $Q' \not\sqsubseteq Q$. □

We show the following complexity result.

Theorem 5.6. *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.* □

Proof: *The problem is in co-NP.* Let Q and Q' be two c-XQueries$^{\neg}$. We describe a non-deterministic polynomial algorithm. We guess a kernel canonical answer CA and a possible partition of the node variables in CA. In the three cases stated above, it takes polynomial time to decide whether $IQ_{CA} \preceq IQ'_{CA}$. Hence, containment of d-XQueries is in co-NP.

The problem is co-NP hard. Follow from the result of testing containment of acyclic queries with negation. □

5.3 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 5.7. *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 4.7 still apply. $\square$

Proof: By applying the technique in [30], given a c-XQuery$^{//}$, we guess a kernel canonical answer CA, and an indexed query IQ_{CA} (by guessing the number of edges for each descendant edge). Checking simulation with consideration of descendant edges takes polynomial time. The problem is coNP-hard following that containment of such XPath expressions is already coNP-hard. $\square$

5.4 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 5.8. *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.* $\square$

Proof: *c-XQueries$^=$ with maximal fanout 1:* Testing containment can be reduced to query simulation in polynomial time, so it is NP-complete.

c-XQueries$^=$ with arbitrary fanout but fixed depth: We guess a kernel canonical answer CA, and decide $IQ_{CA} \not\sqsubseteq IQ'_{CA}$ by a co-NP oracle. So the problem is in Π_p^2. Π_p^2 hardness is proved by reduction from the $\forall\exists$-CNF problem.

Let ψ be a 3-CNF formula with variables $x_1, \ldots, x_n, y_1, \ldots, y_p$ and clauses $c_1, \ldots, c_m$. The $\forall\exists$-CNF problem asks whether for each truth assignment ν to $\bar{x}$, there exists a truth assignment to $\bar{y}$ that satisfies ψ.

We construct two queries cc-XQueries, Q and Q', s.t. ψ is satisfiable iff $Q \not\sqsubseteq Q'$. We show that for each truth assignment to $\bar{x}$, ψ is satisfied by an assignment ν to $\bar{y}$, iff we can construct from ν a canonical answer CA of Q, such that CA is also a canonical answer of Q', DB'_{CA} and DB_{CA} satisfy $DB'_{CA} \sqsubseteq DB_{CA}$. Hence, the proof follows from Theorem 4.5(3).

Both Q and Q' have two levels: a top-level query block, and $2n$ children blocks. In each query, the top level block returns the tag u; for each variable $x_i, i \in [1, n]$, there are two children query blocks on the second level, returning v_i and w_i respectively. Hence, Q and Q' have exactly the same canonical answers.

In query Q, there are three groups of conjuncts. Group A contains $7m + 2n + 2p + 1$ conjuncts in the top-level query block. There is one conjunct $\Re aV$. For each clause $c_j, j \in [1, m]$, there are seven conjuncts: $\Re d_j C_j^k, k \in [1, 7]$. For each variable $X_i, i \in [1, n]$, there are two conjuncts Vx_iX_i and $Vx_iX'_i$. For each variable $Y_i, i \in [1, p]$, there are two conjuncts Vy_iY_i and $Vy_iY'_i$.

In group B, there are $21m$ conjuncts. For each clause $c_j, j \in [1, m]$, there are seven satisfying assignments. In the top-level query block, if y_i is assigned *true* in the $k, k \in [1, 7]$-th assignment, there is a conjunct $C_k sY_i$ in the top-level query block; if y_i is assigned $false$, there is a conjunct $C_k sY'_i$ in the top-level query block. In addition, if x_i is assigned *true* in the k-th assignment, there is a conjunct $C_k sX_i$ in the query block that returns v_i; if if x_i is assigned $false$ in the k-th assignment, there is a conjunct $C_k sX'_i$ in the query block that returns w_i.

In group C, there are $n(4m + n + p)$ conjuncts in total. For each variable $x_l, l \in [1, n]$, there are $4m + p$ conjuncts in the top-level query block. For each clause $c_j, j \in [1, m]$, there is a conjunct $\Re d_j B_j^l$. For each variable $y_i, i \in [1, p]$, there is a conjunct $Vy_iY_i^l$. If variable $y_i, i \in [1, p]$ occurs in clause $c_j, j \in [1, m]$, there is a conjunct $B_j^l sY_i^l$. If variable $x_i, i \in [1, n]$ occurs in clause $c_j, j \in [1, m]$, there is a conjunct $B_j^l sX_i^l$. There are n conjuncts in the second-level query block. For each variable $x_i, x \in [1, n-1]$, there is a conjunct $Vx_iX_i^l$ in the query block returning v_i. Besides, there is a conjunct $Vx_nX_n^l$ in the query block returning w_i.

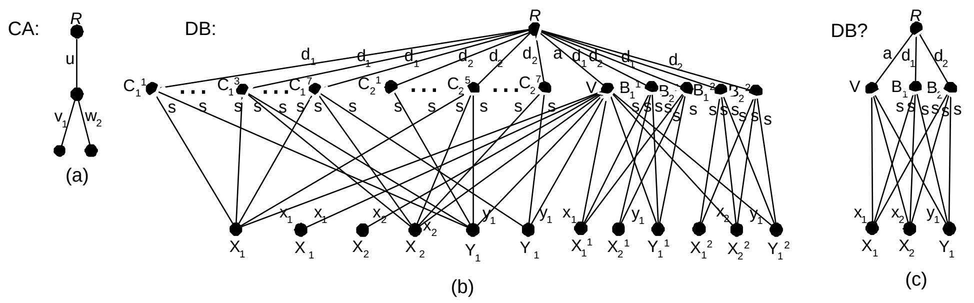


Figure 8: (a) An example canonical answer CA of Q that corresponds to an assignment ν in which $x_1 = true, x_2 = false$; (b) Q and Q''s canonical databases for CA.

Now we consider query Q'. In the top-level query block, there is one conjunct $\Re aV$. Besides, for each variable $X_i, i \in [1, n]$, there is a conjunct Vx_iX_i. For each variable $Y_i, i \in [1, p]$, there is a conjunct Vy_iY_i. For each clause $c_j, j \in [1, m]$, there is a conjunct $\Re d_jB_j$. If variable y_i occurs in clause c_j, there is a conjunct C_jtY_i. In the second-level query blocks, if variable x_i occurs in clause c_j, there is a conjunct C_jsX_i in the query block that returns v_i and the one that returns w_i.

As an example, consider the formula $\psi = (x_1 \vee x_2 \vee \neg y_1) \wedge (\neg x_1 \vee x_2 \vee y_1)$. We build the following two queries.

```
Q:  for $\Re aV, Vx_1X_1, Vx_1X_1', Vx_2X_2, Vx_2X_2', Vy_1Y_1, Vy_1Y_1', Vy_1Y_1^1, Vy_1Y_1^2,$
        $\Re d_1C_1^1, \cdots, \Re d_1C_1^7, \Re d_2C_2^1, \cdots, \Re d_2C_2^7,$
        $C_1^1tY_1, C_1^2tY_1, C_1^3tY_1, C_1^4tY_1', C_1^5tY_1, C_1^6tY_1', C_1^7tY_1',$
        $C_2^1tY_1, C_2^2tY_1, C_2^3tY_1, C_2^4tY_1', C_2^5tY_1, C_2^6tY_1', C_2^7tY_1',$
        $\Re d_1B_1^1, B_1^1sX_1^1, B_1^1sX_2^1, B_1^1sY_1^1, \Re d_2B_2^1, B_2^1sX_1^1, B_2^1sX_2^1, B_2^1sY_1^1,$
        $\Re d_1B_1^2, B_1^2sX_1^2, B_1^2sX_2^2, B_1^2sY_1^2, \Re d_2B_2^2, B_2^2sX_1^2, B_2^2sX_2^2, B_2^2sY_1^2$ return
    [u, {
        for $C_1^1sX_1, C_1^3sX_1, C_1^4sX_1, C_1^7sX_1, C_2^2sX_1, C_2^5sX_1, C_2^6sX_1, Vx_1X_1^1$ return
        [$v_1$,{}]
        for $C_1^2sX_1', C_1^5sX_1', C_1^6sX_1', C_2^1sX_1', C_2^3sX_1', C_2^4sX_1', C_2^7sX_1, Vx_2X_2^1$ return
        [$w_1$,{}]
        for $C_1^1sX_2, C_1^2sX_2, C_1^4sX_2, C_1^6sX_2, C_2^1sX_2, C_2^2sX_2, C_2^4sX_2, C_2^6sX_2, Vx_1X_1^2$ return
        [$v_2$,{}]
        for $C_1^3sX_2', C_1^5sX_2', C_1^7sX_2', C_2^3sX_2', C_2^5sX_2', C_2^7sX_2', Vx_2X_2^2$ return
        [$w_2$,{}]
    }]
Q':  for $\Re aV, Vx_1X_1, Vx_2X_2, Vy_1Y_1, \Re d_1C_1, \Re d_2C_2, C_1sY_1, C_2sY_1$ return
    [u, {
        for $C_1sX_1, C_2sX_1$ return
        [$v_1$,{}]
        for $C_1sX_1, C_2sX_1$ return
        [$w_1$,{}]
        for $C_1sX_2, C_2sX_2$ return
        [$v_2$,{}]
        for $C_1sX_2, C_2sX_2$ return
        [$w_2$,{}]
    }]
```

We construct CA from ν, as follows. Consider a two-level canonical answer, which has a single node with incoming edge labeled u on the first level. There is a node with incoming edge labeled v_i, iff $\nu(x_i) = true$; there is a node with incoming edge labeled w_i, iff $\nu(x_i) = false$. Based on this, we partition these canonical answers into three classes. If both v_i and w_i occur in CA, or neither v_i or w_i occurs in CA, we say the canonical answer corresponds to an *invalid* assignment. The former is called a *conflict* assignment, and the latter is called a *missing* assignment. Otherwise, the canonical answer must correspond to a *valid* assignment. If given the assignment, there does not exist a satisfying assignment for ψ, we say the canonical answer corresponds to a *bad* assignment. If there exists a satisfying assignment for ψ, we say the canonical answer corresponds to a *good* assignment. In our example, a canonical answer that corresponds to the assignment $x_1 = true, x_2 = false$ is shown in Figure 8(a).

Constructing Q and Q' takes polynomial time. Now we show that $Q \sqsubseteq Q'$ iff ψ is satisfiable given any assignment for $\bar{x}$.

if We first discuss two-level canonical answers CA of Q that correspond to a valid assignment ν. Since given any assignment for $\bar{x}$, there exists a satisfying assignment μ for ψ. So CA corresponds to a good assignment. There exists a graph embedding from DB'_{CA} to DB_{CA}, sending V in DB'_{CA} to V in DB_{CA}, sending each $C_i, i \in [1, m]$ to a $C_i^j, j \in [1, 7]$, sending each $X_i, i \in [1, n]$ to X_i if $\nu(x_i) = true$ or X'_i otherwise, and sending each $Y_i, i \in [1, p]$ to Y_i if $\mu(y_i) = true$ or Y'_i otherwise. Thus, $DB'_{CA} \sqsubseteq DB_{CA}$.

To illustrate, the canonical answer in Figure 8(a) corresponds to a good assignment for the given example. The two canonical databases, DB_{CA} and DB'_{CA}, are shown in Figure 8(b). There is a graph embedding sending $V, C_1, C_2, X_1, X_2, Y_1$ in DB'_{CA} to $V, C_1^3, C_2^5, X_1, X'_2, Y_1$ respectively.

If a two-level canonical answers CA of Q corresponds to an invalid assignment, there are two cases. In case CA corresponds to a conflicting assignment, where both v_l and $w_l, l \in [1, n]$ occur in CA. Consider the subgraph containing nodes $V, C_j^l, j \in [1, m], X_i^l, i \in [1, n], Y_k^l, k \in [1, p]$ and associated edges in DB_{CA}. It's trivial that there exists a graph embedding from DB'_{CA} to this subgraph. Thus, $DB'_{CA} \sqsubseteq DB_{CA}$.

In case CA corresponds to a missing assignment, where neither v_l nor $w_l, l \in [1, n]$ occurs in CA. In DB'_{CA} there does not exist any edge from a node representing a clause to a node representing a quantitative variable. Consider the subgraph containing nodes $V, C_j^l, j \in [1, m], Y_k^l, k \in [1, p], X_i, i \in [1, n]$ and associated edges in DB_{CA}. It's trivial that there exists a graph embedding from DB'_{CA} to this subgraph. Thus, $DB'_{CA} \sqsubseteq DB_{CA}$.

Finally, it's easy to verify that for the empty canonical answer CA_0 and the one-level canonical answer CA_1 of Q, we have $DB'_{CA_0} \sqsubseteq DB_{CA_0}, DB'_{CA_1} \sqsubseteq DB_{CA_1}$. This proves $Q \sqsubseteq Q'$.

only if Assume ψ is not satisfiable. Then there must be an assignment ν, s.t. there does not exist a satisfying assignment μ for ψ. The canonical answer CA of Q for ν corresponds to a bad assignment. Trivially, there does not exist a graph embedding from DB'_{CA} to DB_{CA}. So $DB'_{CA} \not\sqsubseteq DB_{CA}$, and so $Q' \not\sqsubseteq Q$. □

5.5 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 [36] it is shown that containment of cyclic queries with arithmetic comparisons is Π_2^p-complete. We show the following. Similar to the case of containment checking for queries with negations, every canonical answer may correspond to several indexed queries. We need to check containment on each of them. Given a canonical answer, we construct indexed queries in three steps: (1) construct a *model indexed query* as in Definition 4.4 ignoring arithmetic comparison predicates; (2) Consider all possible partitions of tag variables in the model indexed query. For each one, consider tag variables in the same group as having the same value, assign order to different variable groups and also tag constants occurring in the queries, resulting in an indexed query.

For query containment $Q \sqsubseteq Q'$ to hold, it must be the case that every indexed conjunctive query IQ_{CA} of Q either violates the negation predicates in Q, or satisfies $IQ_{CA} \preceq IQ'_{CA}$. We show the following:

Theorem 5.9. *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.* □

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						

Proof: The lower bound follows from Theorem 5.8. Now we prove the upper bound. We guess a kernel canonical answer CA, a tag partition, and an order assignment; then we decide $IQ_{CA} \not\preceq IQ'_{CA}$ by a co-NP oracle. So the problem is in Π_p^2. □

6 Conclusions

We considered the complexity of query containment 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 PTIME results in some cases. The main open gap in the complexity analysis is the case of queries with arbitrary nesting depth.

In addition to closing that gap, we are also interested in the complexity of containment when the schema of the input documents is known (hence possibly reducing the number of canonical answers for consideration). We are also building on our containment results to develop algorithms for answering queries using views, where both the queries and the views contain nesting.

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] N. Bidoit. The verso algebra or how to answer queries with fewer joins. *Journal of Computer and System Sciencees*, 35(3):321–364, 1987.

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

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

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

[9] D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In *Proceedings of KR*, pages 2–13, 1998.

[10] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *the 9th Annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[11] 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.

[12] S. Chaudhuri and M. Vardi. Optimizing real conjunctive queries. In *Proc. of PODS*, pages 59–70, Washington D.C., 1993.

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

[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 *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1999.

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

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

[18] C. A. Gunter and D. S. Scott. *Handbook of theoretical computer science: algorithms and complexity*, volume B. MIT Press, 1990.

[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. Halevy, Z. Ives, I. Tatarinov, and P. Mork. Piazza: Data management infrastructure for semantic web applications. In *Proc. of the Int. WWW Conf.*, 2003.

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

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

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

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

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

[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*, pages 171–181, Dublin, Ireland, 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*, Washington D.C., 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] J. D. Ullman. Information integration using logical views. In *the International Conference on Database Theory*, 1997.

[36] 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.

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

[38] X. Zhang and M. Z. Ozsoyoglu. On efficient reasoning with implication constraints. In *Proc. of DOOD*, pages 236–252, 1993.