Conjunctive Query Containment over Trees[⋆]

Henrik Björklund, Wim Martens, and Thomas Schwentick

University of Dortmund

Abstract. The complexity of containment and satisfiability of conjunctive queries over finite, unranked, labeled trees is studied with respect to the axes *Child*, *NextSibling*, their transitive and reflexive closures, and *Following*. For the containment problem a trichotomy is presented, classifying the problems as in PTIME, coNP-complete, or Π_2^P-complete. For the satisfiability problem most problems are classified as either in PTIME or NP-complete.

1 Introduction

Conjunctive query containment for relational databases is one of the most thoroughly investigated problems in database theory. It is known to be essentially equivalent to conjunctive query evaluation and to Constraint Satisfaction in AI [9]. From the database point of view, the importance of conjunctive queries on relational structures lies in the fact that they correspond to the most widely used queries in practice. More precisely, they correspond to the select-from-where queries from SQL that only use "and" as a Boolean connective.

Recently, conjunctive queries over trees also attracted quite some attention [7]. It is somewhat surprising that they have not been studied earlier, as they arise very naturally in various settings, such as data extraction and integration, computational linguistics, and dominance constraints [7]. Moreover, unary and binary conjunctive queries over trees form a very natural fragment of XPath 2.0 [1], and therefore also of XQuery [2]. Indeed, unary and binary conjunctive queries over trees correspond to Core XPath without *negation* and *union* (see, e.g., [6]), but with *path intersection*, as introduced in XPath 2.0 (see, e.g., [8,13]). Gottlob et al. already showed that unary conjunctive queries over trees can be translated to XPath 1.0 queries, albeit with an exponential blow-up [7], and the above-mentioned Core XPath queries with path intersection can be translated into conjunctive queries by identifying variables. Hence, our complexity upper bounds transfer to positive Core XPath expressions with path intersection, but without union.

In this paper, we consider conjunctive query containment on trees. We mainly focus on Boolean containment of conjunctive queries, i.e., given two conjunctive queries P and Q, is $L(P) \subseteq L(Q)$, where $L(P)$ (resp., $L(Q)$) denotes the set of trees on which P (resp., Q) has a non-empty output. Conjunctive query containment over trees is a problem that needs to be solved for conjunctive query

⋆ This work was supported by the DFG Grant SCHW678/3-1.

Table 1. Complexities of Conjunctive Query Containment

	Child	*Child*$^+$	*Child**	*NextSibling*	*NextSibling*$^+$	*NextSibling**	*Following*
Child	in P	Π_2^P	Π_2^P	coNP	coNP	coNP	Π_2^P
Child$^+$		coNP	coNP	Π_2^P	Π_2^P	Π_2^P	Π_2^P
Child*			coNP	Π_2^P	Π_2^P	Π_2^P	Π_2^P
NextSibling				in P	coNP	coNP	Π_2^P
NextSibling$^+$					coNP	coNP	Π_2^P
NextSibling*						coNP	Π_2^P
Following							coNP

optimization. The latter is, for instance, important for XQuery engines, but is also relevant in the other settings mentioned above. Moreover, conjunctive query *satisfiability*, which we also study and which is a simplified form of containment, needs to be solved if one wants to decide well-definedness for important XQuery fragments [14]. There is a further relevant setting in which the set of trees under consideration is restricted by a schema and the containment question is asked relative to this schema. We give a brief overview of our results.

Containment. We obtain a similar classification as Gottlob et al. [7]. The most essential differences are that the PTIME membership results for conjunctive query evaluation translate to coNP membership results for containment and that NP-completeness results for evaluation translate to Π_2^P-completeness results for containment. The former translation is easy to obtain due to a polynomial size witness property for counter examples (Lemma 8). For the latter translation, we build on some of the NP lower bound reductions by Gottlob et al. for our Π_2^P lower bound proofs. They had to be significantly adapted, however, as unlike in the relational setting, conjunctive query containment on trees cannot be reduced in a straightforward manner to conjunctive query evaluation on a canonical model. Most of our complexity results on conjunctive query containment are summarized in Table 1. From the above mentioned polynomial size witness property and the results by Gottlob et al. [7], we can also conclude that containment is in coNP for the fragments CQ(*Child*, *NextSibling*, *NextSibling**, *NextSibling*$^+$), CQ(*Child**, *Child*$^+$), and CQ(*Following*). Combined with the results from the table, this gives us a complete trichotomy of the complexity of conjunctive query containment with respect to all sets of axes we consider.

Unfortunately, as we can see from the table, conjunctive query containment on trees is quite a hard problem. We only identify two tractable fragments, that is, CQ(*NextSibling*) and CQ(*Child*). For the latter fragment, PTIME membership is already non-trivial. All other combinations of axes are at least coNP-hard.

Satisfiability. Conjunctive query satisfiability can be seen as a simplification of the containment problem. Indeed, Q is satisfiable if and only if $L(Q) \nsubseteq L(\text{false})$. Our results on satisfiability are summarized in Table 2. Interestingly, we see here that the dichotomy drawn by the evaluation and the containment problem shifts.

Table 2. Complexities of Conjunctive Query Satisfiability

	Child	*Child*$^+$	*Child**	*NextSibling*	*NextSibling*$^+$	*NextSibling**	*Following*
Child	in P	NP [8]	NP	in P	in P	in P	NP
Child$^+$		in P	in P	?	?	?	?
Child*			in P	?	?	?	?
NextSibling				in P	NP	NP	NP
NextSibling$^+$					in P	in P	in P
NextSibling*						in P	in P
Following							in P

For the satisfiability problem, we obtain significantly more tractable fragments than for the containment problem. Some cases, however, still remain NP-hard.

We note that the NP lower bound for satisfiability of CQ(*Child*,*Child*$^+$) is already obtained by Hidders [8]. We prove it in an alternative manner in Theorem 26.

Containment with respect to a schema. It turns out that the complexity of the containment problem is (presumably) much higher if it is posed relative to a given schema which restricts the set of trees under consideration. A similar effect was observed before for XPath query containment [11]. More concretely, we show that deciding whether a conjunctive query only using *Child*-axes returns a non-empty output on each tree defined by a DTD is EXPTIME-hard. In fact, the conjunctive query in our proof can even be expressed as an XPath query using wildcards, predicates, and the axes *Child* and *Descendant*, thereby obtaining that XPath containment w.r.t. a DTD is EXPTIME-complete for XPath queries using *Child*, *Descendant*, predicates, and wildcards.

Related Work. Most of the related work has already been mentioned. We note, however, that conjunctive query containment has also been investigated for object-oriented database systems [3]. In particular, it is shown that conjunctive query containment is Π_2^P-complete. The classes of conjunctive queries studied in [3] are, however, incomparable to ours.

2 Preliminaries

2.1 Trees

By Σ we always denote a fixed but infinite set of labels. For a finite set S, we denote by $|S|$ the number of elements of S. The trees we consider are rooted, ordered, finite, labeled, unranked trees, which are directed from the root downwards. That is, we consider trees with a finite number of nodes and in which nodes can have arbitrarily many children. We view a tree t as a relational structure over a finite number of unary labeling relations $a(\cdot)$, where each $a \in \Sigma$, and binary relations *Child*$(\cdot,\cdot)$ and *NextSibling*$(\cdot,\cdot)$. Here, $a(u)$ expresses that u is a node with label a, and *Child*(u, v) (respectively, *NextSibling*(u, v)) expresses that v is a child (respectively, next sibling) of u.

Notice that, in contrast to standard practice, we have an infinite set of labels from which our (finite) trees can choose. This reflects how trees occur in an XML-context: an XML tree is a finite structure, but there is no restriction on how it should be labeled (if no schema is provided).

In addition to *Child* and *NextSibling*, we use their transitive closures (denoted Child^+ and $\mathit{NextSibling}^+$) and their transitive and reflexive closures (denoted Child^* and $\mathit{NextSibling}^*$). We also use the *Following*-relation, which is inspired by XPath [4] and defined as

$$\mathit{Following}(u, v) = \exists x \exists y\, \mathit{Child}^*(x, u) \wedge \mathit{NextSibling}^+(x, y) \wedge \mathit{Child}^*(y, v).$$

We denote the set of nodes of a tree t by $\mathrm{Nodes}(t)$. We define the *size of* t, denoted by $|t|$, as the number of nodes of t. We refer to the above mentioned binary relations as *axes.*

2.2 Conjunctive Queries

Let $X = \{x, y, z, \dots\}$ be a set of variables. A *conjunctive query* (CQ) over alphabet Σ is a positive existential first-order formula without disjunction over a finite set of unary predicates $a(x)$ where each $a \in \Sigma$, and the binary predicates *Child*, Child^+, Child^*, *NextSibling*, $\mathit{NextSibling}^+$, $\mathit{NextSibling}^*$, and *Following*. In this paper, we will mainly focus on Boolean satisfaction of conjunctive queries. We will therefore consider conjunctive queries without free variables. As our conjunctive queries do not contain free variables, we sometimes omit the existential quantifiers to simplify notation. For a conjunctive query Q, we denote the set of variables appearing in Q by $\mathrm{Var}(Q)$. We use $\mathrm{CQ}(R_1, \dots, R_k)$ or $\mathrm{CQ}(\mathcal{R})$ (where $\mathcal{R} = \{R_1, \dots, R_k\}$) to denote the fragment of CQs that uses only the unary alphabet predicates and the binary predicates $R_1, \dots, R_k$. We use the terminology on valuations of a query and query graphs from Gottlob et al. [7].

Definition 1. Let Q be a conjunctive query, and t a tree. A *valuation* of Q on t is a total function $\theta : \mathrm{Var}(Q) \to \mathrm{Nodes}(t)$. A valuation is a *satisfaction* if it satisfies the query, that is, if every atom of Q is satisfied by the assignment. A tree t *models* Q ($t \models Q$) if there is a satisfaction of Q on t. The language $L(Q)$ of Q is the set of all trees that model Q.

We say that a tree t is a *minimal model* of Q if $t \models Q$ and the number of nodes in t is minimal among all trees in $L(Q)$.

The following example illustrates a conjunctive query.

Example 2. Consider the conjunctive query $Q = \mathit{Child}^+(x_1, x_2) \wedge \mathit{Child}^+(x_2, x_4) \wedge \mathit{Child}^+(x_1, x_3) \wedge \mathit{Child}^+(x_3, x_4) \wedge a(x_1) \wedge b(x_2) \wedge c(x_3) \wedge d(x_4) \wedge e(x_5)$. A tree t models Q if t has an a-labeled node u with a d-labeled descendant v such that the path from u to v contains a b-labeled node and a c-labeled node (in arbitrary order). Moreover, t must contain an e-labeled node somewhere.

Definition 3. Let Q be a conjunctive query over Σ with variables $\mathrm{Var}(Q)$. The query graph Q is the directed multigraph $G_Q = (V, E)$ with edge labels and node labels such that $V = \mathrm{Var}(Q)$, node x is labeled a if and only if $a(x)$ is an atom in Q; and E contains the labeled directed edge $x \xrightarrow{R} y$ if and only if $R(x, y)$ is an atom in Q.

We assume familiarity with standard graph-related terminology such as *reachability*, *connected components*, etc. Subgraphs of G_Q correspond to subqueries of Q. We will sometimes slightly abuse the terminology by using graph-related concepts when talking about queries. Thus "variable x is reachable from variable y in Q" means that x is reachable from y in G_Q. Similarly, "maximal connected component of Q" means a subquery corresponding to a maximal connected component of G_Q.

For ease of readability, we often represent queries (or their query graphs) graphically. For example, the rightmost picture in Figure 2 represents the query

$$\mathit{Child}(x_1, x_2) \wedge \mathit{Child}(x_2, x_3) \wedge \mathit{Child}(x_2, x_4) \wedge b(x_3) \wedge c(x_4).$$

The following decision problems for conjunctive queries are the main topic of interest for this paper.

Definition 4

- Containment: Given two conjunctive queries P and Q, is $L(P) \subseteq L(Q)$?
- Satisfiability: Given a conjunctive query Q, is $L(Q) \neq \emptyset$?

The above problems are in a sense both instances of the containment problem. That is, satisfiability for Q is testing whether $L(Q) \not\subseteq L(\text{false})$.

As mentioned above, we consider conjunctive queries without free variables. This means that we only look at whether a tree models the query or not, and not at the whole set of satisfactions. One can also consider *k-ary conjunctive queries*, i.e., CQs with k free variables, returning a k-ary relation when evaluated on a tree. For two k-ary queries P and Q, *P is contained in Q* if, for every tree t, the relation returned by P is a subset of the relation returned by Q. Using a result of Miklau and Suciu [10], this problem reduces to containment for Boolean queries for all fragments that include the *Child*-axis. For instance, consider the left query $P(x_1, x_2, x_3)$ in Figure 1. By introducing, for each free variable x_i, a new variable x_i' and adding the atoms $\mathit{Child}(x_i, x_i') \wedge X_i(x_i')$ to the query, where X_i is a new label, the query P', depicted on the right of Figure 1, is obtained. It is now easy to see that, for two queries $P(\overline{x})$ and $Q(\overline{x})$[1] with k free variables, P is contained in Q if and only if $L(P') \subseteq L(Q')$, where P' and Q' are obtained by adding the atoms $\mathit{Child}(x_i, x_i') \wedge X_i(x_i')$ to P and Q, respectively. Indeed, the proof is analogous to the one in [10]. For satisfiability, it is of course immediate that the complexities are the same for 0-ary and k-ary queries.

2.3 Basic Properties

If t and t' are trees, h is a function from t to t', and $\mathcal{R}$ is a set of binary relations, we say that h is an *$\mathcal{R}$-homomorphism* if $h(u)$ is defined for every node u in t,

[1] We can assume w.l.o.g. that the free variables are the same in P and Q.

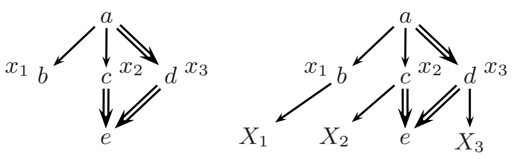

Fig. 1. How to reduce from k-ary queries to 0-ary queries

$a(u)$ in t implies $a(h(u))$ in t', for each $a \in \Sigma$, and $R(u,v)$ holds in t implies that $R(h(u),h(v))$ holds in t', for each $R \in \mathcal{R}$.

Observation 5. Let t be a tree and let $Q \in CQ(\mathcal{R})$ be a query such that $t \models Q$. If t' is a tree and there exists an $\mathcal{R}$-homomorphism $h : t \to t'$, then $t' \models Q$.

Observation 6. Conjunctive queries are monotonous, i.e., if $t \models Q$, for a tree t and a CQ Q, then $t' \models Q$ for all trees t' for which $t \subseteq t'$.

3 Containment

When we investigate whether query P is contained in query Q, i.e., $L(P) \subseteq L(Q)$, we will always assume that the graph of Q has only one maximal connected component. This is because P is contained in Q if and only if P is contained in every subquery of Q that corresponds to a maximal connected component.

3.1 PTIME Upper Bounds

Theorem 7. *Containment is in PTIME for CQ(Child) and CQ(NextSibling).*

Proof (Sketch). The proof for CQ(*NextSibling*) is straightforward: for testing whether $L(P) \subseteq L(Q)$, one can start by simplifying both queries, by applying the chase for the relation $\mathit{NextSibling}(A,B)$ with functional dependencies $A \to B$ and $B \to A$ (i.e., $\mathit{NextSibling}(x,x_1) \wedge \mathit{NextSibling}(x,x_2)$ or $\mathit{NextSibling}(x_1,x) \wedge \mathit{NextSibling}(x_2,x)$ should not occur). Consequently, we have to test whether the chased query for Q can be embedded into the chased query for P.

The proof for CQ(*Child*) is tedious and too intricate to illustrate the main idea here. We just want to point out that the problem is not trivial. A naive algorithm would try to find an embedding of Q into P and accept iff it can be found. However, Figure 2 illustrates that not finding an embedding of Q into P does not imply that $L(P) \not\subseteq L(Q)$. □

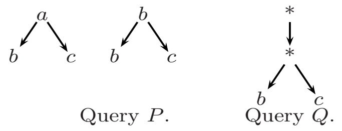

Fig. 2. Example for which $L(P) \subseteq L(Q)$ but Q cannot obviously be embedded into P. Every arrow denotes a *Child*-axis.

$$\begin{tikzcd}
 & a_1^1 \arrow[r] & a_1^2 \arrow[r] & \ldots \arrow[r] & a_1^{n_1} \arrow[dr] & \\
\# \arrow[ur] \arrow[dr] & \vdots & & & \vdots & \# \\
 & a_m^1 \arrow[r] & a_m^2 \arrow[r] & \ldots \arrow[r] & a_m^{n_m} \arrow[ur] &
\end{tikzcd}$$

Fig. 3. Structure of query P in the proof of Theorem 10

3.2 coNP and Π_2^P Upper Bounds

We first show that if CQ P is not contained in CQ Q, then there is a polynomial size witness tree.

Lemma 8. *Let P and Q be conjunctive queries. If $L(P) \not\subseteq L(Q)$ then there exists a tree t such that $t \models P$, $t \not\models Q$, and $|t| \leq 2 \cdot |\mathit{Var}(P)| \cdot (|\mathit{Var}(Q)| + 4)$.*

The above lemma puts conjunctive query containment in Π_2^P. Indeed, for testing whether $L(P) \not\subseteq L(Q)$, the algorithm would guess a tree t_{small} of size at most $2 \cdot |\text{Var}(P)| \cdot (|\text{Var}(Q)| + 4)$, test in NP whether $t_{\text{small}} \models P$ and test in coNP whether $t_{\text{small}} \not\models Q$. As Gottlob et al. showed that conjunctive query evaluation is in PTIME for CQ(*Child*, *NextSibling*, *NextSibling**, *NextSibling*$^+$), CQ(*Child**,*Child*$^+$), and CQ(*Following*), the above algorithm gives us a coNP upper bound for containment for these fragments. We can therefore state the following theorem.

Theorem 9.

1. *Containment is in Π_2^P for CQs.*
2. *Containment is in coNP for CQ(Child*,Child$^+$), CQ(Following), and CQ(Child,NextSibling,NextSibling*,NextSibling$^+$).*

3.3 coNP Lower Bounds

For the coNP lower bounds, we will reduce from the complement of the Shortest Common Supersequence problem; or the Shortest Common Superstring problem, both of which are known to be NP-complete [12,5]. The Shortest Common Supersequence (respectively, Shortest Common Superstring) problem asks, given a set of strings S, and an integer k, whether there exists a string of length at most k which is a supersequence (respectively, superstring) of each string in S. Here, s is a supersequence of s_0 if s_0 can by obtained by deleting symbols from s, and s is a superstring of s_0 if s_0 can be obtained by deleting a prefix and a postfix of s.

Theorem 10. *Containment is coNP-hard for CQ(NextSibling$^+$), CQ(NextSibling*), CQ(Child$^+$), CQ(Child*), and CQ(Following).*

Proof. All cases can be proved by a reduction from Shortest Common Supersequence. Thereto, let S and k be an instance of Shortest Common Supersequence. We now define conjunctive queries P and Q such that $P \not\subseteq Q$ if and only if there exists a shortest common supersequence for S of length at most k. Thereto, let $S = \{s_1, \ldots, s_m\}$ where, for each $i = 1, \ldots, m$, $s_i = a_i^1 \cdots a_i^{n_i}$. Let $\#$ be a symbol not occurring in any string in S.

We first show how the proof works for *NextSibling*$^+$. The query P is defined as in Figure 3, where each arrow represents a *NextSibling*$^+$-axis and $\#$ and each a_i^j is a Σ-symbol. The query Q now essentially states that each tree must have a string of siblings with at least $k + 1 + 2$ different nodes. Formally, we define Q as

$$\mathit{NextSibling}^+(x_1, x_2) \wedge \cdots \wedge \mathit{NextSibling}^+(x_{k+2}, x_{k+3}).$$

It is not difficult to see that $P \not\subseteq Q$ if and only if there exists a shortest common supersequence for S of length at most k. The proofs for *Child*$^+$and *Following* are completely analogous. For *Child**and *NextSibling**, we need to insert dummy $\#$-symbols between all a_i^j labels in P, and adapt the query Q accordingly. □

The proof of the next theorem is in the same lines as the previous one, but this time we reduce from the Shortest Common Superstring problem. The essential difference is that P now does not contain the leftmost and rightmost $\#$-labeled symbol in Figure 3, the arrows in Figure 3 now denote *NextSibling*-axes, and that all the a_j^i-labeled nodes are connected to a common parent by *Child*-axes.

Theorem 11. *Containment is coNP-hard for CQ(Child,NextSibling).*

3.4 Π_2^P Lower Bounds

The Π_2^P lower bounds in this section will all be obtained by a reduction from $\forall\exists$ positive 1-in-3 SAT, which is formally defined as follows. A set $C_1, \ldots, C_m$ of clauses is given, each of which has three Boolean variables from $\{x_1, \ldots, x_{n_x}\} \uplus \{y_1, \ldots, y_{n_y}\}$. No variable is negated. The question is whether, for every truth assignment for $\{x_1, \ldots, x_{n_x}\}$, there exists a truth assignment for $\{y_1, \ldots, y_{n_y}\}$ such that each C_i contains precisely one true variable.

The proof of the following lemma is analogous to the proof showing that positive 1-in-3 SAT is NP-complete.

Lemma 12. *$\forall\exists$ positive 1-in-3 SAT is Π_2^P-complete.*

Theorem 13. *Containment is Π_2^P-complete for CQ(Child, Child$^+$) and CQ(Child, Child*).*

Proof. We present a proof for CQ(*Child*, *Child*$^+$) and discuss in the end how to adapt it for CQ(*Child*, *Child**).

The proof is an adaptation of a proof by Gottlob et al., showing that the query complexity of evaluation for CQ(*Child*, *Child*$^+$) is NP-hard [7]. We reduce from $\forall\exists$ positive 1-in-3 SAT, which is Π_2^P-complete according to Lemma 12.

For the purposes of this proof, we will assume that each tree node can carry multiple labels. It can be modified to work for the standard definition of labeled trees, where each node has only one label.

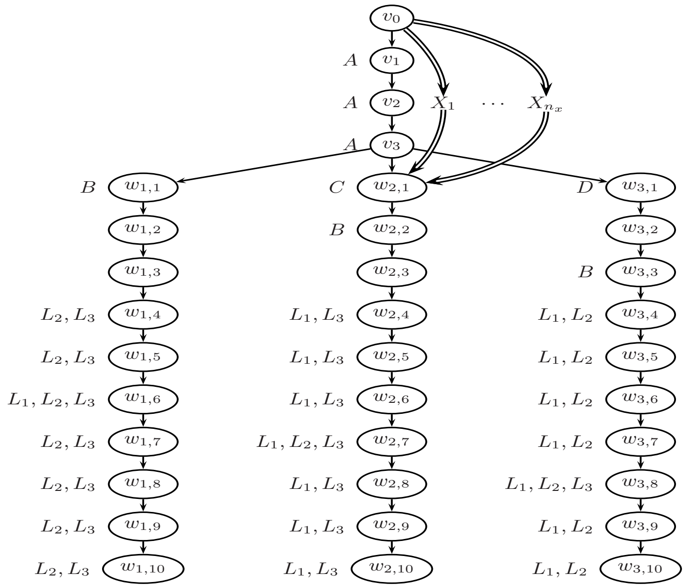

Fig. 4. Illustration of the definition of query P in the proof of Theorem 13

Let $\forall\overline{x}\exists\overline{y}C_1,\ldots,C_m$ be an instance of $\forall\exists$ positive 1-in-3 SAT, where $\overline{x} = \{x_1,\ldots,x_{n_x}\}$ and $\overline{y} = \{y_1,\ldots,y_{n_y}\}$. We may assume that no clause contains a particular literal more than once. Let Φ denote the formula

$$\forall\overline{x}\exists\overline{y}C_1,\ldots,C_m,C_{m+1},\ldots C_{m+n_x}.$$

Here, for each $i = 1,\ldots,n_x$, C_{m+i} denotes the clause (y_i', x_i, y_i''), where y_i' and y_i'' are new existentially quantified variables. It is easy to see that there is solution for the original formula if and only if there is one for Φ.

Let query P be defined as in Figure 4, where single lines represent the *Child* axis, double lines represent the *Child*$^+$axis, and the symbols outside of nodes, as well as $X_1,\ldots X_{n_x}$, are labels.

For the query Q, we introduce variables a_i, b_i for each $i = 1,\ldots,m+n_x$ and in addition a variable $c_{k,l,i,j}$ whenever the k-th literal of C_i coincides with the l-th literal of C_j $(1 \le j \le m+n_x,\ i \ne j,\ 1 \le k,l \le 3)$.

The query Q consists of the following atoms:

- for each $i = 1,\ldots,m+n_x$, $A(a_i) \wedge B(b_i) \wedge \mathit{Child}^3(a_i,b_i)$;
- for each variable $c_{k,l,i,j}$, $L_k(c_{k,l,i,j}) \wedge \mathit{Child}^+(b_i,c_{k,l,i,j}) \wedge \mathit{Child}^{8+k+l}(a_j,c_{k,l,i,j})$; and,
- for each $i = m+1,\ldots,m+n_x$, $X_{i-m}(a_i)$.

This concludes the reduction for CQ(*Child*, *Child*$^+$). For CQ(*Child*,*Child**) we replace each pair of atoms $\mathit{Child}^+(v_0,X_i)$, $\mathit{Child}^+(X_i,w_{2,1})$ of P (for $1 \le i \le n_x$) with the pair $\mathit{Child}^*(v_1,X_i)$, $\mathit{Child}^*(X_i,v_3)$. In Q, we can simply replace *Child*$^+$ by *Child**. □

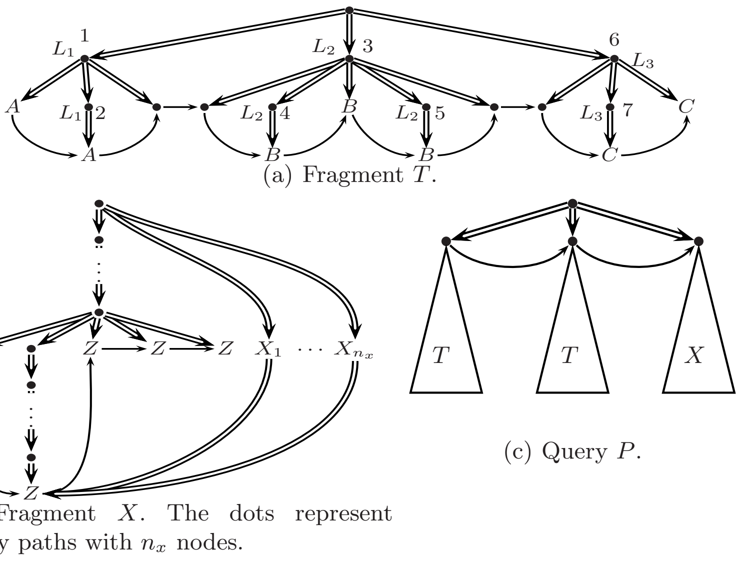

Fig. 5. Definition of query P in the proof of Theorem 15

Theorem 14. *Containment is Π_2^P-hard for CQ(Child, Following).*

Proof. We adapt the proof of Theorem 13 by simulating *Child*$^+$with *Child* and *Following*. To this end, we begin by equipping each of the variables u in query P defined in Figure 4 that has an outgoing *Child*$^+$-axes by two "dummy" children z_1 and z_2. These new variables are used nowhere else, and get the new label #. Now, whenever *Child*$^+(u, v)$ is used in one of the queries, we can replace it by

$$\mathit{Child}(u, z_1) \wedge \mathit{Child}(u, z_2) \wedge \mathit{Following}(z_1, v) \wedge \mathit{Following}(v, z_2).$$

It is now enough to note that all variables in the queries P and Q that have no specified label are required by the queries to have children. Thus none of them can bind to a node in one of the minimal models of the modified P query that is labeled by #. □

Theorem 15. *Containment is Π_2^P-hard for CQ(Child$^+$,Following).*

Proof (Sketch). Let $\forall\overline{x}\exists\overline{y}C_1, \ldots, C_m$ be an instance of $\forall\exists$ positive 1-in-3 SAT. Let $\overline{x} = \{x_1, \ldots, x_{n_x}\}$ and let $\overline{y} = \{y_1, \ldots, y_{n_y}\}$. We can assume that no clause contains a particular literal more than once.

We construct two queries, P and Q, over the labeling alphabet $\{A, B, C, L_1, L_2, L_3, X_1, \ldots, X_{n_x}, Z\}$ such that $P \subseteq Q$ if and only if $\forall\overline{x}\exists\overline{y}C_1, \ldots, C_m$ has a solution. The current proof builds further on a proof by Gottlob et al. that shows that the query complexity of evaluation for CQ(*Child*$^+$,*Following*) is NP-hard [7].

The construction of query P is illustrated in Figure 5. Here, every double-lined edge represents a *Child*$^+$-axis and every directed edge represents a *Following*-axis. For improved readability, we adopt the terminology of the proof by Gottlob

$$\begin{tikzcd}
L_1 \arrow[r, "F^2"'] \arrow[rr, bend left, "F^7"] \arrow[d, Rightarrow] & L_2 \arrow[r, "F^2"'] \arrow[d, Rightarrow] & L_3 \arrow[d, Rightarrow] \\
A \arrow[r, "F^4"'] & B \arrow[r, "F^4"'] & C
\end{tikzcd}$$

(a) The 1-in-3 gadget.

$$\begin{tikzcd}
L_1 \arrow[r, "F^{21}"] \arrow[d, Rightarrow] & X_i \arrow[d, Rightarrow] \\
\bullet \arrow[r, "F^{23}"] & Z
\end{tikzcd}
\qquad
\begin{tikzcd}
L_2 \arrow[r, "F^{16}"] \arrow[d, Rightarrow] & X_i \arrow[d, Rightarrow] \\
\bullet \arrow[r, "F^{20}"] & Z
\end{tikzcd}
\qquad
\begin{tikzcd}
L_3 \arrow[r, "F^{13}"] \arrow[d, Rightarrow] & X_i \arrow[d, Rightarrow] \\
\bullet \arrow[r, "F^{15}"] & Z
\end{tikzcd}$$

(b) The X-variable gadgets: varX$(1,i)$ (left), varX$(2,i)$ (middle), and varX$(3,i)$ (right).

Fig. 6. Gadgets for the definition of query Q in the proof of Theorem 15

et al.. That is, we will refer to the nodes labeled L_1, L_2, and L_3 in the *1-in-3 gadget* from Figure 6(a) by v_1, v_2, and v_3, respectively. Moreover, we annotate the query fragment T in Figure 5(a) with numbers from 1 to 7. We call the node 1 (resp., 3, 6) the *topmost position* of variable v_1 (resp., v_2, v_3).

Let $t_{\min}$ be a minimal model of fragment T from Figure 5(a). That is, $t_{\min}$ is essentially shaped as the structure given by the *Child*$^+$ axes in T. Gottlob et al. show that the following observation holds.

Observation 16 ([7]). Every satisfaction θ of the 1-in-3 gadget on t_{min} maps exactly one of the variables v_1, v_2, and v_3 to its topmost position.

Given a clause C, we interpret a satisfaction θ in which variable v_k is mapped to its topmost position as the selection of the k-th literal from C to be true. Hence, the 1-in-3 gadget would ensure that, on $t_{\min}$, exactly one variable of clause C is selected and becomes true.

We now define the query P as in Figure 5(c). That is, P contains two copies of the fragment T, followed by a copy of the X-fragment from Figure 5(b). The ordering between the subqueries of P is enforced by *Following*-axes: the root of T's left copy has a *Following*-axis to the root of T's right copy, and root of T's right copy has a *Following*-axis to the root of the X-fragment.

Intuitively, the purposes of the different parts of the query P are as follows. The left copy of the T-fragment in P, together with the 1-in-3 gadget, is used to verify that the truth assignments we consider for $\overline{x}$ and $\overline{y}$ actually make one literal per clause of $\forall\overline{x}\exists\overline{y}C_1,\ldots,C_m$ true. The second copy of T in P is needed to ensure consistency of variable assignments between clauses: if we pick a variable to be true in one clause, that variable must be true in all clauses. Finally, the fragment X is used in P to generate all possible truth assignments for the $\overline{x}$-variables. Roughly, we interpret x_i as "false" if X_i appears in the upper unary path with n_x nodes and as "true" if X_i appears in the lower unary path with n_x nodes in Figure 5(b).

The query Q is defined much like the query in the proof of Gottlob et al., with the essential difference that we have to transfer the variable assignment that is generated in the X-fragment of P to the matching of L_1, L_2, and L_3 of the 1-in-3 gadget of Q onto the subtrees that satisfy the two copies of T in P. This will be taken care of by the X-assignment gadgets in Q, which are illustrated in Figure 6(b). □

Theorem 17. *Containment is Π_2^P-hard for CQ(Child*, Following).*

Proof. The proof for this case can be obtained from the proof of Theorem 15 by ensuring that, for each occurrence of *Child*$^+(x, y)$, x and y bear a different alphabet label in P. □

As *Following* can be defined in terms of *Child** and *NextSibling*$^+$, we immediately have the following corollary.

Corollary 18. *Containment is Π_2^P-hard for CQ(Child*, NextSibling$^+$).*

Theorem 19. *Containment is Π_2^P-hard for*
(1) CQ(Child, NextSibling),*
(2) CQ(Child,NextSibling*),*
(3) CQ(Child$^+$,NextSibling),
(4) CQ(Child$^+$,NextSibling$^+$), and
(5) CQ(Child$^+$,NextSibling).*

Proof. For each of these fragments, the proofs of Theorems 15 and 17 can be adapted by the same methods as in the article by Gottlob et al. [7]. For the fragments (2)–(5), we also need to adapt the query P, such that P accepts trees in which the T-fragments have the shape from the proof by Gottlob et al. This is, however, straightforward for each of the fragments. □

Theorem 20. *Containment is Π_2^P-hard for CQ(Following,NextSibling).*

The proof uses a modified version of the reduction from Theorem 15.

Theorem 21. *Containment is Π_2^P-hard for CQ(Following, NextSibling$^+$) and CQ(Following, NextSibling*).*

4 Satisfiability

We first note that a conjunctive query Q is satisfiable if and only if all its maximal connected components are satisfiable. We therefore assume in our proofs that Q has only one maximal connected component.

Proposition 22. *Satisfiability for CQs is in NP.*

Proof. It is easy to see that if a CQ is satisfiable, then it is satisfiable in a linear size tree. Indeed, let Q be a CQ and let t be a tree satisfying Q under valuation θ. Now let t' be the tree that

– contains the nodes of t onto which variables are matched by θ;
– contains, for each pair of variables $x \neq y$, the least common ancestor of $\theta(x)$ and $\theta(y)$;

– contains no other nodes; and
– preserves the descendant relation and document order (i.e., depth-first-left-to-right order) from t.

It is easy to see that t' contains less than $2 \cdot |\mathrm{Var}(Q)|$ nodes and that t' models Q. Thus we can guess this tree, guess a satisfaction, and verify in polynomial time that all atoms are satisfied. □

4.1 PTIME Upper Bounds

Theorem 23. *Satisfiability is in PTIME for CQ(Child) and CQ(NextSibling).*

Proof. First, we apply the chase on the relations in Q, i.e., we compute equivalence classes $[x]$ of variables such that $[x]$ is the set of variables that must be mapped to the same tree node as x by any satisfaction of Q. For $Q \in CQ(\mathit{Child})$, we start with one class for each variable in $\mathrm{Var}(Q)$, and iteratively merge classes $[x]$ and $[y]$ if there are $x' \in [x]$, $y' \in [y]$, and a variable z such that there is are paths from x' to z and from y' to z in G_Q, both of equal length. For $Q \in CQ(\mathit{NextSibling})$ we do the same, with the addition that we also merge classes $[x]$ and $[y]$ if there are $x' \in [x]$, $y' \in [y]$, and z such that there are equal length paths from z to x' and from z to y'.

Once these classes are computed, we test whether Q contains a cycle on equivalence classes. If Q contains a cycle, it is unsatisfiable. Otherwise, we check whether there exist a class containing two variables x, y and $a \neq b$ such that $a(x)$ and $b(y)$ are both atoms of Q. If this is the case, Q is unsatisfiable. Otherwise it is satisfiable. □

Theorem 24. *Satisfiability is in PTIME for CQ(NextSibling$^+$,NextSibling*, Following) and CQ(Child$^+$,Child*).*

Proof. As in the proof of Theorem 23 we first check for cycles. However, unlike for Theorem 23, a query may have cycles of Child^*(resp., $\mathit{NextSibling}^*$) axes and still be satisfiable. On such cycles, there can be no variables x, y such that $a(x)$ and $b(y)$ are atoms, for $a \neq b$. We start by removing such (allowed) cycles by identifying all variables on the cycle. In the remainder of the proof, we assume that the query is cycle free.

For CQ($\mathit{NextSibling}^+$,$\mathit{NextSibling}^*$,$\mathit{Following}$), we argue that if Q is satisfiable, then there is a tree t and a satisfaction θ for Q on t such that θ assigns all variables of Q to nodes of t which are all siblings of one another (i.e., in the same *siblinghood*). As a first step we note that, if Q is satisfiable, then Q', obtained by replacing all $\mathit{NextSibling}^*$-atoms of Q by $\mathit{NextSibling}^+$-atoms is also satisfiable. Indeed, if θ is a satisfaction of Q on tree t, $\mathit{NextSibling}^*(x, y)$ is an atom of Q, and $\theta(x) = \theta(y)$, we can modify t by inserting a new node between $\theta(x)$ and its left sibling (or at the beginning of the siblinghood if there is no left sibling), and modify θ by assigning x to the new node. After doing this for all such pairs x, y, the modified θ is a satisfaction of both Q and Q'.

Next, we note that any acyclic query Q in CQ(*NextSibling*$^+$,*Following*) induces a strict partial order on the variables. A topological sorting according to this partial order gives us a string of variables such that if $\mathit{NextSibling}^+(x,y)$ or $\mathit{Following}(x,y)$ is an atom of Q, then x appears before y in the string. From such a string it is easy to construct a tree with a siblinghood that satisfies Q.

For CQ(*Child*$^+$, *Child**) we use the same arguments as for CQ(*NextSibling*$^+$, *NextSibling**, *Following*), except that instead of a siblinghood we use a tree that does not branch. □

Theorem 25. *Satisfiability is in PTIME for CQ(Child, NextSibling) and CQ(Child, NextSibling*$^+$*, NextSibling***).*

4.2 NP Lower Bounds

Theorem 26. *Satisfiability is NP-hard for*

(1) CQ(Child, Child$^+$*),*
(2) CQ(Child,Child**),*
(3) CQ(NextSibling,NextSibling$^+$*),*
(4) CQ(NextSibling,NextSibling**),*
(5) CQ(NextSibling,Following), and
(6) CQ(Child,Following).

The proof uses reductions from Shortest Common Supersequence.

5 Containment with Respect to a DTD

We abstract from Document Type Definitions (DTDs) as follows:

Definition 27. A *Document Type Definition (DTD)* is a triple $D = (\Sigma_f, d, s_d)$ where Σ_f is a finite subset of Σ, d is a function that maps Σ_f-symbols to regular expressions over Σ_f and $s_d \in \Sigma_f$ is the start symbol.

For ease of notation, we denote by $\mathrm{lab}^t(u)$ the Σ-symbol a such that $a(u)$ appears in t. A tree t then *satisfies* D if *(i)* $\mathrm{lab}^t(u) = s_d$ for the root u of t and, *(ii)* for every $u \in \mathrm{Nodes}(t)$ with n children $u_1, \ldots, u_n$ from left to right, $\mathrm{lab}^t(u_1)\cdots\mathrm{lab}^t(u_n) \in L(d(\mathrm{lab}^t(u)))$. By $L(d)$ we denote the set of trees satisfying d.

The main problem of interest of this section is *validity of a conjunctive query Q w.r.t. a DTD D*, that is, is $L(D) \subseteq L(Q)$? Notice that here, validity with respect to a DTD can also be seen as a form of containment of conjunctive queries with respect to a DTD. That is, $L(D) \subseteq L(Q)$ if and only if $(L(D) \cap L(\mathrm{true})) \subseteq (L(D) \cap L(Q))$.

Theorem 28. *Validity with respect to DTDs is EXPTIME-hard for CQ(Child).*

6 Conclusions

We have determined the complexity of the containment problem for all sets of axes built from *Child*, *NextSibling*, their transitive, respectively reflexive and transitive, closures, and *Following*. The complexity of the satisfiability problem was pinpointed for most sets, but the cases involving transitive closures of *Child* and *NextSibling* (which we believe will be quite similar) are still open.

All these results were obtained in a schema-less setting. Since XML processing is mostly done with respect to a schema, this is far from the complete picture. As we show in Section 5, the presence of a schema dramatically increases the complexity. We have preliminary results for some combinations of schemas and axes, and intend to study this subject in more detail.

References

1. Berglund, A., Boag, S., Chamberlin, D., Fernández, M.F., Kay, M., Robie, J., Siméon, J.: XML Path Language (XPath) 2.0. Technical report, World Wide Web Consortium (January 2007), `http://www.w3.org/TR/xpath20/`
2. Boag, S., Chamberlin, D., Fernández, M.F., Florescu, D., Robie, J., Siméon, J.: Xquery 1.0: An XML query language. Technical report, World Wide Web Consortium (January 2007), `http://www.w3.org/TR/xquery/`
3. Chan, E.P.F., van der Meyden, R.: Containment and optimization of object-preserving conjunctive queries. Siam J. on Computing 29(4), 1371–1400 (2000)
4. Clark, J., DeRose, S.: XML Path Language (XPath) version 1.0. Technical report, World Wide Web Consortium (1999), `http://www.w3.org/TR/xpath/`
5. Gallant, J., Maier, D., Storer, J.A.: On finding minimal length superstrings. JCSS 20(1), 50–58 (1980)
6. Gottlob, G., Koch, C., Pichler, R., Segoufin, L.: The complexity of XPath query evaluation and XML typing. Journal of the ACM 52(2), 284–335 (2005)
7. Gottlob, G., Koch, C., Schulz, K.U.: Conjunctive queries over trees. Journal of the ACM 53(2), 238–272 (2006)
8. Hidders, J.: Satisfiability of XPath expressions. In: Lausen, G., Suciu, D. (eds.) DBPL 2003. LNCS, vol. 2921, pp. 21–36. Springer, Heidelberg (2004)
9. Kolaitis, P.G., Vardi, M.Y.: Conjunctive-query containment and constraint satisfaction. JCSS 61(2), 302–332 (2000)
10. Miklau, G., Suciu, D.: Containment and equivalence for a fragment of XPath. Journal of the ACM 51(1), 2–45 (2004)
11. Neven, F., Schwentick, T.: On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. Logical Methods in Computer Science 2(3) (2006)
12. Räihä, K.J., Ukkonen, E.: The shortest common supersequence problem over binary alphabet is NP-complete. TCS 16(2), 187–198 (1981)
13. ten Cate, B., Lutz, C.: The complexity of query containment in expressive fragments of XPath 2.0. In: PODS 2007. 26th International Symposium on Principles of Database Systems, pp. 73–82 (2007)
14. Vansummeren, S.: On deciding well-definedness for query languages on trees. Journal of the ACM 54(4) (2007)