Journal of Computer and System Sciences 77 (2011) 450–472

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Conjunctive query containment over trees ☆

Henrik Björklund, Wim Martens *,[1], Thomas Schwentick

Technical University of Dortmund, Germany

ARTICLE INFO

Article history:
Received 31 January 2008
Received in revised form 11 December 2008
Available online 24 April 2010

Keywords:
Data management
XML
Query optimization
Logic

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 [12]. From the database point of view, the importance of conjunctive queries on relational structures lies in the fact that they are 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 have also been studied over tree structures [10]. 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 [10]. Moreover, unary and binary conjunctive queries over trees form a very natural fragment of XPath 2.0 [2], and therefore also of XQuery [5]. Indeed, unary and binary conjunctive queries over trees are closely related to Core XPath without *negation* and *union* (see, e.g., [9]), but with *path intersection*, as introduced in XPath 2.0 (see, e.g., [11,16]). 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 [10], 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 nonempty output. Conjunctive query containment over trees is a problem that needs to be solved for conjunctive query 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 frag-

☆ This work was supported by the DFG Grant SCHW678/3-1. The present paper is the full version of Ref. Björklund et al. (2007) [3], which appeared in the Symposium on Data Base Programming Languages 2007.
* Corresponding author.
E-mail addresses: henrik.bjoerklund@udo.edu (H. Björklund), wim.martens@udo.edu (W. Martens), thomas.schwentick@udo.edu (T. Schwentick).
[1] Supported by a grant of the North-Rhine Westfalian Academy of Sciences.

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2010.04.005

Table 1
Complexities of conjunctive query containment. All coNP and Π_2^P results are completeness results.

	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

Table 2
Complexities of conjunctive query satisfiability. All NP-results are completeness results.

	Child	*Child*$^+$	*Child**	*NextSibling*	*NextSibling*$^+$	*NextSibling**	*Following*
Child	in P	NP [11]	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

ments [17]. 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 [4]. We give a brief overview of our results.

Containment. We obtain a similar classification as Gottlob et al. [10]. 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 1). 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. [10], we can also conclude that containment is in Π_2^P for conjunctive queries using all axes, and that it is in coNP for the fragments using the axes {*Child*, *NextSibling*, *NextSibling**, *NextSibling*$^+$}, {*Child**, *Child*$^+$}, and {*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 subsets of the 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, conjunctive queries using either only the *NextSibling*-axis, or only the *Child*-axis. 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 from the evaluation and containment problems shifts. 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 conjunctive queries with {*Child*, *Child*$^+$} was already obtained by Hidders [11]. We give an alternative proof (Theorem 23).

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 [6]. In particular, it is shown that conjunctive query containment is Π_2^P-complete. The classes of conjunctive queries studied in [6] are, however, incomparable to ours. Also, Arenas and Libkin study data exchange for tree patterns, which form a subset of conjunctive queries, and also prove results for satisfiability w.r.t. a DTD [1]. This is an issue that we also studied in [4], but do not consider in the current paper.

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.

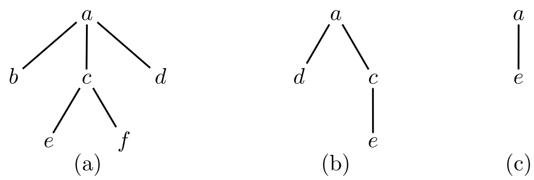


Fig. 1. Examples of $\mathcal{R}$-subtrees.

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. We assume that each node in a tree bears precisely one label, i.e., for each u, there is precisely one $a \in \Sigma$ such that $a(u)$ holds in t.

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 $NextSibling^+$) and their transitive and reflexive closures (denoted $Child^*$ and $NextSibling^*$). We also use the *Following*-relation, which is inspired by XPath [7] and defined as

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

We denote the set of nodes of a tree t by $Nodes(t)$. 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*.

A tree t' is a *subtree* of a tree t if t' is a tree and a substructure of t. In other words, t' is connected and the relations in t' are subsets of the *Child*- and *NextSibling* relations in t. Furthermore, the labels of nodes are the same in t and t'. Our definition of subtree implies that, when u is a next sibling of v in a subtree t' of t, u also has to be a next sibling of v in t itself. As this is a quite severe restriction, we sometimes also want to work with a more flexible notion of subtrees. Therefore, for a set $\mathcal{R}$ of axes, we say that t' is an $\mathcal{R}$-subtree of t if t' is a tree and, for each axis R in $\mathcal{R}$, $R(x, y)$ in t' implies that $R(x, y)$ holds in t. Just as in normal subtrees, t' preserves the labels of t. Intuitively, in an $\mathcal{R}$ subtree, we want all relations in $\mathcal{R}$ to be preserved. The "standard" notion of subtree defined above corresponds with a $\{Child, NextSibling\}$-subtree. As an example of $\mathcal{R}$-subtrees, consider Fig. 1. Fig. 1(a) contains a tree t. Fig. 1(b) contains a $\{Child\}$-subtree of t, and Fig. 1(c) a $\{Child^+\}$-subtree. Notice that siblings in the $\{Child\}$-subtree are also siblings in t, but their ordering can be different: the d-labeled node can come before the c-labeled node whereas they were ordered the other way around in t.

2.2. Conjunctive queries

Let $X = \{x, y, z, \ldots\}$ 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*, $NextSibling^+$, $NextSibling^*$, 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 queries do not contain free variables, we often 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 $CQ(R_1, \ldots, R_k)$ or $CQ(\mathcal{R})$ (where $\mathcal{R} = \{R_1, \ldots, R_k\}$) to denote the fragment of CQs that uses only the unary alphabet predicates and the binary predicates $R_1, \ldots, R_k$. We use the terminology on valuations of a query and query graphs from Gottlob et al. [10].

Definition 1. Let Q be a conjunctive query, and t a tree. A *valuation* of Q on t is a total function $\theta : \mathrm{Var}(Q) \to Nodes(t)$. 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 (denoted $t \models Q$) 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 $L(Q)$.

For readability, we often represent queries graphically. We often omit the variable names of the queries in the figures and only present their Σ-symbols. If a variable has no associated Σ-symbol, we denote this by a wildcard symbol "$*$". The following example presents such an illustration of a conjunctive query.

Example 1. Consider the conjunctive query $Q = Child^+(x_1, x_2) \wedge Child^+(x_2, x_4) \wedge Child^+(x_1, x_3) \wedge Child^+(x_3, x_4) \wedge Child^+(x_5, x_6) \wedge a(x_1) \wedge b(x_2) \wedge c(x_3) \wedge d(x_5) \wedge e(x_6)$. Fig. 2(a) depicts query Q. Any tree t that models Q must have an a-labeled node u with a 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 d-labeled node with an e-labeled child somewhere. An example of such a tree is in Fig. 2(b).

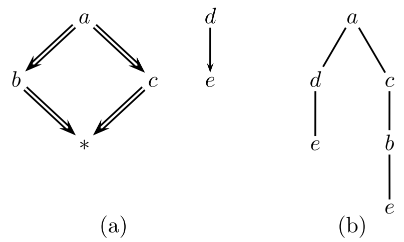


Fig. 2. Graphical representation of the query in Example 1 and a tree modeling it. Double arrows represent *Child*$^+$ relations and single arrows represent *Child* relations.

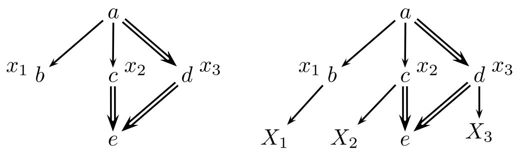


Fig. 3. How to reduce from k-ary queries to 0-ary queries.

Definition 2. Let Q be a conjunctive query over Σ with variables $\mathrm{Var}(Q)$. The query graph Q is the directed edge- and node-labeled multigraph $G_Q = (N, E)$ with nodes N and edges E such that $N = \mathrm{Var}(Q)$, 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.

Sometimes, we use the notation $R^i(x, y)$, where R is an axis and $i \in \mathbb{N}$. This means that y can be reached from x using i steps of R, and is shorthand for $R(x, x_1) \wedge R(x_1, x_2) \wedge \cdots \wedge R(x_{i-1}, y)$, where $x_1, \ldots, x_{i-1}$ are variables that do not appear anywhere else in the query.

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

Definition 3.

- 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 $L(Q) \nsubseteq L(\text{false})$.

For the containment problem, many of our algorithms will search for a tree t such that $t \in L(P) - L(Q)$. If $t \in L(P) - L(Q)$, we call t a *counterexample*. Similarly, for the satisfiability problem, we will often search for a tree $t \in L(Q)$, which we call a *witness*.

Boolean versus k-ary queries. As mentioned above, we consider conjunctive queries without free variables. The result of evaluating such a query on a tree is therefore Boolean. In general 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 [13], 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 Fig. 3. By introducing, for each free variable x_i, a new variable x_i' and adding the atoms $Child(x_i, x_i') \wedge X_i(x_i')$ to the query, where X_i is a new label, the query $P'(x_1, x_2, x_3)$, depicted on the right of Fig. 3, is obtained. It is now easy[2] to see that, for two queries $P(\bar{x})$ and $Q(\bar{x})$[3] 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 $Child(x_i, x_i') \wedge X_i(x_i')$ to P and Q, respectively. For satisfiability, it is of course immediate that the complexities are the same for 0-ary and k-ary queries.

[2] The proof is analogous to the one of Proposition 1 in [13].

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

2.3. Basic properties

In this section we list a few basic properties of conjunctive queries which are quite well known and easy to prove. We use them further on in our proofs. 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, $a(u)$ 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 1. 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 2. Conjunctive queries are monotonous. More precisely, let Q be a $CQ(\mathcal{R})$ and let $t \models Q$. Then $t' \models Q$ for all trees t' for which t is an $\mathcal{R}$-subtree of t'.

For the next observation, we extend the notion of $\mathcal{R}$-homomorphisms to queries. That is, if P and Q are in $CQ(\mathcal{R})$, we say that $h : \mathrm{Var}(P) \to \mathrm{Var}(Q)$ is an $\mathcal{R}$-homomorphism from P to Q if h is total, $a(x)$ in P implies that $a(h(x))$ in Q for each $a \in \Sigma$, and $R(x_1, x_2)$ in P implies that $R(h(x_1), h(x_2))$ occurs in Q, for each $R \in \mathcal{R}$.

Observation 3. Let P and Q be in $CQ(\mathcal{R})$. If there exists a homomorphism from Q to P, then $L(P) \subseteq L(Q)$.

As we will see in the proof of Theorem 6, the other direction of Observation 3 does not always hold.

2.4. Chasing queries in $CQ(Child)$ and $CQ(NextSibling)$

In some of our upper bound proofs using only *Child* or only *NextSibling*, will use *the chase*, which is a well-know technique in database theory. Let Q be a query in $CQ(Child)$ or $CQ(NextSibling)$. The idea behind the chase is that we compute equivalence classes $[x]$ of variables such that $[x]$ is the maximal set of variables such that for any $t \in L(Q)$ and any satisfaction θ for Q on t, we must have $\theta(y) = \theta(x)$ for all $y \in [x]$. We do this as follows:

- For $Q \in CQ(Child)$, 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 both $Child(x', z)$ and $Child(y', z)$ are atoms of Q.
- For $Q \in CQ(NextSibling)$ 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 $NextSibling(z, x')$ and $NextSibling(z, y')$ are both atoms of Q.

Once we have computed the equivalence classes, i.e., when no more classes can be merged using the rules above, we rewrite Q, obtaining a new query $\mathrm{chase}(Q)$, which is the result of applying the chase to Q. This is done by creating a new variable for each class and replacing each occurrence of a variable in Q with the variable representing its class. Notice that, in each of these cases, $\mathrm{chase}(Q)$ can be computed in polynomial time.

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.

Observation 4. Let P and Q be CQs and let $Q_1, \ldots, Q_k$ be the maximal connected components of Q. Then $L(P) \subseteq L(Q)$ if and only if $L(P) \subseteq L(Q_i)$ for all $i \in \{1, \ldots, k\}$.

3.1. PTIME upper bounds

Theorem 5. *Containment is in PTIME for $CQ(NextSibling)$.*

Proof. For testing whether $L(P) \subseteq L(Q)$, where $P, Q \in CQ(NextSibling)$, we first test that both queries are satisfiable. This can be done in polynomial time by Theorem 20. If P is unsatisfiable, containment trivially holds. If Q is unsatisfiable while P is satisfiable, containment fails. As the chase can be computed in polynomial time for P and Q (see Section 2.4), we can assume that $P = \mathrm{chase}(P)$ and $Q = \mathrm{chase}(Q)$. Hence, each query is a collection of *linear* maximal connected subqueries, that is, there are no variables $x \neq y \neq z$ such that both $NextSibling(x, y)$ and $NextSibling(x, z)$ or $NextSibling(y, x)$ and $NextSibling(z, x)$ are atoms. Furthermore, by Observation 4, we can assume that Q has only one maximal connected component.

We now claim that containment holds if and only if there is a homomorphism from Q to P. Since the queries are linear, testing if there is a homomorphism can be done in polynomial time. If there is such a homomorphism, containment trivially holds. If not, we construct a counterexample tree $t \in L(P) - L(Q)$ as follows. Let $P_1, \ldots, P_k$ be the maximal connected

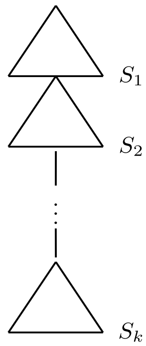


Fig. 4. The tree construction for *CQ*(*NextSibling*) containment.

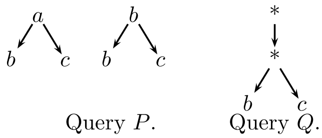


Fig. 5. Example for which $L(P) \subseteq L(Q)$, but there is no homomorphism from Q to P. Each arrow denotes a *Child*-axis.

components of P. Each such P_i has variables $x^i_1, \dots, x^i_{n_i}$, binary atoms $\mathit{NextSibling}(x^i_j, x^i_{j+1})$ for each $j \in \{1, \dots, n_i - 1\}$, and a number of unary atoms. With each P_i we associate a string S_i of length n_i. Position j of S_i has label $a \in \Sigma$ if $a(x^i_j)$ is an atom of P_i. If there is no such atom, position j gets label #, where $\# \in \Sigma$ is a symbol that occurs in neither P nor Q.

The tree t is depicted in Fig. 4 and has levels $0, 1, \dots, k$. On each level, except level k, there is exactly one node that has children—all the others are leaves. On level 0, there is only the root, which has label #. All nodes on level i, for $i \in \{1, \dots, k\}$ are children of the sole non-leaf node on level $i - 1$. Level i has n_i nodes, which are labeled, from left to right, with the symbols of S_i.

Clearly, t satisfies P. We claim that t does not satisfy Q. Suppose, towards a contradiction, that there would be a satisfaction of Q on t. Then this satisfaction would induce a homomorphism from Q to P, which we assumed not to be the case. □

Theorem 6. *Containment is in PTIME for CQ*(*Child*).

Proof. The proof for *CQ*(*Child*) is considerably more involved than the one for *CQ*(*NextSibling*). A naive algorithm would try to find a homomorphism of Q into P and accept if and only if it can be found. However, Fig. 5 illustrates that not finding a homomorphism from Q into P does not imply that $L(P) \not\subseteq L(Q)$. Indeed, there is no homomorphism from Q to P, but $L(P) \subseteq L(Q)$.

Let P and Q be two queries in *CQ*(*Child*). We want to decide whether $L(P) \subseteq L(Q)$. First, we check if the two queries are satisfiable. This can be done in polynomial time by Theorem 20. If at least one of the queries is not satisfiable, we already have our answer, so we assume in the remainder of the proof that both are satisfiable.

Any satisfiable query in *CQ*(*Child*) can be transformed in polynomial time into an equivalent one that is *tree shaped*, i.e., such that there are no variables $x \neq y \neq z$ such that both $\mathit{Child}(x, z)$ and $\mathit{Child}(y, z)$ are atoms of the query. This is achieved by applying the chase procedure (Section 2.4). Since P and Q have already been checked for satisfiability and the chase can be computed in polynomial time, we may assume that both P and Q are tree shaped and contain no cycles.

In the remainder of this proof, we use the notions of *distance* between nodes and the *length of a path* in a tree. When $u_1, \dots, u_n$ are nodes in a tree t such that, for every $i = 1, \dots, n - 1$, either $\mathit{Child}(u_i, u_{i+1})$ or $\mathit{Child}(u_{i+1}, u_i)$ holds, then $p = (u_1, u_2), (u_2, u_3), \dots, (u_{n-1}, u_n)$ is a *path* between u_1 and u_n in t. The *length of* p is $n - 1$, which corresponds to the number of edges in p. The *distance* between two nodes u and v in t is the length of the shortest path between u and v.

First, we test whether there is a homomorphism from Q to P. As we can assume that the queries are tree shaped, this can be done in polynomial time; see, e.g., [13]. If there is a homomorphism, we can conclude that $L(P) \subseteq L(Q)$. From now on, we assume that there is no homomorphism from Q to P.

If there is no homomorphism and P has only one maximal connected component, we can conclude that $L(P) \not\subseteq L(Q)$, since none of P's minimal models model Q. However, if there is no homomorphism from Q to P and P has more than one maximal connected component, it is still possible that $L(P) \subseteq L(Q)$ holds, as in the example in Fig. 5.

We try to find a counterexample to containment, that is, a tree that satisfies P but not Q. As usual, by Observation 4, we can assume that Q has only one maximal connected component.

Since Q is tree shaped, it has a unique root variable r_Q. Let $C_1, \dots, C_k$ be the subqueries of Q such that the root r_{C_j} of each C_j is a child of r_Q (i.e., $\mathit{Child}(r_Q, r_{C_j})$ is an atom of Q). Also, let $P_1, \dots, P_m$ be the maximal connected components of

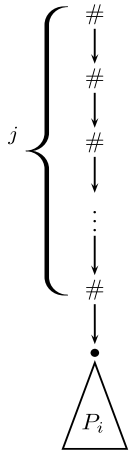


Fig. 6. P_i^j.

P and let the root variable of each P_i be r_{P_i}. If r_Q has a label (i.e., $a(r_Q)$ is an atom of Q for some $a \in \Sigma$), we can easily find a counterexample tree: a root labeled with a new symbol # which has the roots of minimal models for $P_1, \ldots, P_m$ as children. Since there is no homomorphism from Q to P, in particular, there is no homomorphism from Q to P_i for any i. Therefore, this counterexample tree does not model Q. Thus we assume in the following that r_Q has no label.

In the following, we will reason about what criteria a counterexample tree must satisfy, and try to construct one that satisfies them. If we succeed with the construction, it is clear that containment fails. On the other hand, if we find that it is impossible to construct a tree that satisfies the criteria, containment holds.

Let $n = |\mathrm{Var}(Q)|$. For each $j \in \{0, \ldots, n\}$ and $i \in \{1, \ldots, m\}$, let P_i^j be the query obtained by adding new variables $z_1, \ldots, z_j$ to P_i, each of them labeled by the new symbol #, adding the atoms $\mathit{Child}(z_l, z_{l+1})$ for $1 \leqslant l < j$, and $\mathit{Child}(z_j, r_{P_i})$; see Fig. 6. In particular, $P_i^0 = P_i$.

Now, for each $1 \leqslant i \leqslant m$, we define v_i to be the largest number smaller than $n+1$ such that there is no homomorphism from Q to $P_i^{v_i}$. Notice that v_i is well defined since there is no homomorphism from Q to P. Also notice that, if there is no homomorphism from Q to P_i^n, there is no homomorphism from Q to P_i^l for any $l \in \mathbb{N}$, since Q has n variables, is connected, and only uses the *Child* axis.

The intuition behind the above construction is the following. If $v_i < n$, then there is a homomorphism from Q to $P_i^{v_i+1}$. This means that, for any counterexample tree t in $L(P) - L(Q)$ and any satisfaction θ of P on t, the distance from the root of t to $\theta(r_{P_i})$ cannot be larger than v_i. Indeed, since the label # does not occur in Q and there is a homomorphism from Q to $P_i^{v_i+1}$, any tree which has a path of length $v_i + 1$ above a model for P_i also satisfies Q.

For some P_i, we may have $v_i = 0$. This means that there is no homomorphism from Q to $P_i^0 = P_i$, but there is a homomorphism from Q to P_i^1. Using the above arguments, for any counterexample tree t and any satisfaction θ for P on t, the root variable r_{P_i} of P_i must be assigned to the root of t by θ. Let s be the number of maximal connected components P_i of P such that $v_i > 0$. Without loss of generality, we assume that $v_i > 0$ for all i in $\{1, \ldots, s\}$ while $v_j = 0$ for all j in $\{s+1, \ldots, m\}$. If there are two components, P_{i_1} and P_{i_2} such that $i_1, i_2 > s$ and the roots of P_{i_1} and P_{i_2} have different labels, they cannot both be assigned to the root of a counterexample tree. Thus no witness tree exists, and we can conclude that containment holds.

Recall that $C_1, \ldots, C_k$ are the subqueries of Q attached to its root variable. For each $1 \leqslant j \leqslant k$ and each $1 \leqslant i \leqslant s$, we define S_j^i to be the subset of $\{0, \ldots, v_i - 1\}$ such that for each $l \in S_j^i$, there is a homomorphism h from C_j to P_i^l that assigns r_{C_j} to the root variable of P_i^l. The intuition behind this definition is the following. Suppose t is a tree in $L(P)$ and θ is a satisfaction for P on t such that the distance t to $\theta(r_{P_i})$ is d, and $d - 1 \in S_j^i$. Then there is a satisfaction θ_{C_j} for C_j on t such that $\theta_{C_j}(r_{C_j})$ is a child of the root of t.

In the remainder of our procedure, we now iterate over every C_j ($1 \leqslant j \leqslant k$). In each iteration, we will try to construct a counterexample tree for containment. If, for every j, we can construct no such tree, we will prove that $L(P) \subseteq L(Q)$.

To this end, fix a C_j. We will try to construct a tree t_j in $L(P)$ such that

1. every satisfaction for Q on t_j *must assign* r_Q *to the root of* t_j, and
2. there is no satisfaction for Q on t_j that *assigns the root variable of* C_j *to a child of the root of* t_j.

If we can find such a t_j, then containment fails.

For each $1 \leqslant i \leqslant s$, we pick an arbitrary number x_i from $\{0, \ldots, v_i - 1\} - S_j^i$. If, for some i, there is no such number, i.e., $\{0, \ldots, v_i - 1\} = S_j^i$, then we set $x_i = -1$.

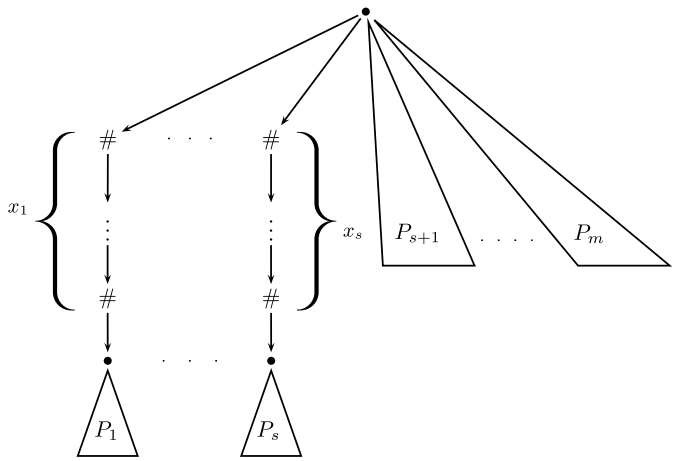


Fig. 7. The construction of t_j, the candidate counterexample tree for C_j.

Given these values $x_1, \dots, x_s$ we try to construct a counterexample tree t_j as follows. For each $1 \leqslant i \leqslant s$ such that $x_i \geqslant 0$, we place the root r_{P_i} of a minimal model for P_i at depth $x_i + 1$ (where 0 is the depth of the root). Between the root and r_{P_i}, we place a non-branching path of nodes labeled #. For the remaining maximal subqueries of P, i.e., the P_i such that $i \in \{s+1, \dots, m\}$ or $x_i = -1$, we take a minimal model of P_i and identify its root with the root of t_j. This may cause a conflict, if two or more of these minimal models already have fixed and different labels. If this is the case, our attempt to construct a counterexample tree for C_j fails and we must iterate to the next C_j. Otherwise, if at least one of them has a fixed root label a, the root of t_j gets label a. If not, then we give the root label #. This construction is depicted in Fig. 7. Clearly, $t_j \in L(P)$. For each P_i such that $i \leqslant s$, the distance from the root to r_{P_i} is smaller than v_i. This means that it is impossible for Q to match along any of the branches from the root, i.e., the root r_Q of Q has to be matched at the root of t_j if it can be matched at all.

We now test whether $t_j \models Q$, and we have found a counterexample tree if $t_j \not\models Q$. If $t_j \models Q$, we have failed to construct a counterexample tree, and we must iterate to the next C_j.

Claim 1. *Let t_j be the tree we have constructed for C_j. If $t_j \models Q$ then, for every tree $t \in L(P)$,*

- *either $t \models Q$, or*
- *there is a satisfaction for C_j on t that assigns r_{C_j} to a child of the root of t.*

Proof. Let θ_Q be a satisfaction for Q on t_j. By construction of t_j, we know that $\theta_Q(r_Q)$ must be the root of t_j. Also, θ_Q must assign all variables of C_j completely within a subtree of t_j corresponding to a minimal model of some P_i with $i > s$ or such that $x_i = -1$. Otherwise, $\theta_Q(r_{C_j})$ would have to be a child of the root of t_j that corresponds to a minimal model of $P_i^{x_i}$, for some $i \leqslant s$ and $x_i \geqslant 0$. But we know that $x_i \notin S_j^i$ for any $i \leqslant s$. Thus there is no $P_i^{x_i}$ such that C_j can be matched in a corresponding minimal model. We conclude that there is a P_i with $i > s$ or $x_i = -1$ such that there is a homomorphism from C_j to P_i that maps r_{C_j} to a child of the root of P_i.

Now consider a tree $t \in L(P)$ and a satisfaction θ_P for P on t. If $\theta_P(r_{P_i})$ is the root of t, then there is a satisfaction for C_j on t that assigns r_{C_j} to a child of the root of t. If, on the other hand, the distance from the root of t to $\theta_P(r_{P_i})$ is $d > 0$, then we have two cases:

1. If $d > v_i$, then there is a satisfaction for Q on t.
2. If $d \leqslant v_i$, then, since we have $x_i = -1$, we know that $d - 1 \in S_j^i$, and again there must be a satisfaction for C_j on t that assigns r_{C_j} to a child of the root of t.

This concludes the proof of Claim 1. □

If we go through all subqueries C_j of Q without being able to successfully construct a witness tree, we argue that containment holds.

Since $t_j \models Q$ for every candidate counterexample t_j, Claim 1 tells us that for every tree $t \in P(t)$ and every $j \in \{1, \dots, k\}$, there is a satisfaction θ_{C_j} for C_j on t that assigns the root of C_j to a child of the root of t.

Since we have assumed that r_Q has no label in Q, we can assign r_Q to the root of t. By combining this assignment with the assignments θ_{C_j} we obtain a satisfaction for Q on t. □

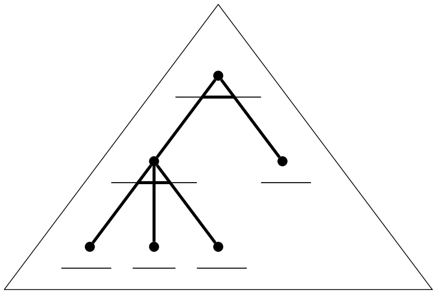


Fig. 8. The construction of t' from t. The dots represent nodes in the set S. A horizontal line under a node represents the children of that node. The bold lines represent the nodes that are kept in t', while everything else is deleted.

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 1. *Let P and Q be conjunctive queries. If $L(P) \nsubseteq L(Q)$ then there exists a tree t such that $t \models P$, $t \not\models Q$, and $|t| \leqslant 2 \cdot |\mathrm{Var}(P)| \cdot 2(|\mathrm{Var}(Q)| + 5)$.*

Proof. Let t be a tree such that $t \models P$ and $t \not\models Q$. Let θ be a satisfaction of P on t, and let $T = \{\theta(x) \mid x \in \mathrm{Var}(P)\}$. Let S be the set of nodes v of t such that v is the least common ancestor of some nonempty subset of T. Further, let R be the set of nodes that belong to a path between two nodes in S.

We will now remove nodes from t to obtain a new tree t'. The construction of t' from t is depicted in Fig. 8 and formally described as follows. We remove all nodes u from t such that

- $u \notin R$, and
- u does not have both an earlier (left) sibling and a later (right) sibling in R, i.e., there do not exist x and y in R with $\mathit{NextSibling}^+(x, u)$ and $\mathit{NextSibling}^+(u, y)$.

Thus we obtain a new tree t'. Clearly, $t' \models P$ and θ is a satisfaction of P on t'. Notice that there is no node u in t such that u is deleted while both its parent and one of its children belong to t'. Also, there is no u in t that is deleted while both its left sibling and its right sibling belong to t'. This means that t' is a $\{\mathit{Child}, \mathit{NextSibling}\}$-subtree of t. Since all other axes are transitive, t' is also an $\mathcal{R}$-subtree of t, where $\mathcal{R}$ is the set of all axes. Hence, by Observation 2, $t' \not\models Q$. Notice that $|S| < 2 \cdot |\mathrm{Var}(P)|$, and that t' only branches at nodes in S.

Now suppose that $|t'| > 2 \cdot |\mathrm{Var}(P)| \cdot 2(|\mathrm{Var}(Q)| + 5)$. We argue that one of the following two cases must occur:

Case 1. There is a pair u, v of nodes in t' from S such that

1. u is an ancestor of v;
2. the path ρ from u to v has length at least $|\mathrm{Var}(Q)| + 4$; and
3. no internal node on ρ belongs to S.

Case 2. There is a pair u, v of nodes in t' such that

1. u and v are siblings, and children from a node in S;
2. there are at least $|\mathrm{Var}(Q)| + 2$ nodes between u and v; and
3. no node between u and v has a descendant in S.

Indeed, towards a contradiction, suppose that none of the two cases applies. We will now count the nodes in t' and show that t' cannot contain more than $2 \cdot |\mathrm{Var}(P)| \cdot 2(|\mathrm{Var}(Q)| + 5)$ nodes, which is a contradiction. First of all, we already noted that S contains at most $2|\mathrm{Var}(P)|$ nodes. Now, to every node u in t' that belongs to S, we will associate two sets N_u^1 and N_u^2, of at most $|\mathrm{Var}(Q)| + 5$ nodes in t'. For each $u \in S$, the sum of the sizes of N_u^1, N_u^2 is at most $2(|\mathrm{Var}(Q)| + 5)$. Furthermore, if we take the union of all N_u^1, N_u^2 over all nodes u in S, we have all the nodes in t'.

Consider, for each node $u \in S$,

- the lowest ancestor u_{anc} of u in S, if it exists. If it does not exist, then u is the root of t'. In this case u_{anc} is undefined, $N_u^1 = \{u\}$, and $N_u^2 = \emptyset$;
- the child $u_{\text{anc-child}}$ of u_{anc} on the path to u;

- the leftmost child $u_{\text{anc-child-right}}$ of u_{anc} to the right of $u_{\text{anc-child}}$ that has a descendant in S, if it exists. Otherwise, $u_{\text{anc-child-right}}$ is undefined and $N_u^2 = \emptyset$.

Let N_u^1 be the set containing u and all nodes between u and u_{anc}. Let N_u^2 be the set of all children of u_{anc} between $u_{\text{anc-child}}$ and $u_{\text{anc-child-right}}$. Observe that

$$\mathit{Nodes}(t') = \bigcup_{u \in S} N_u^1 \cup N_u^2.$$

If Case 1 does not apply, then N_u^1 contains at most $|\mathrm{Var}(Q)| + 4$ nodes. If Case 2 does not apply, then N_u^2 contains at most $|\mathrm{Var}(Q)| + 3$ nodes. Hence, this implies that t' can contain not more than $2|\mathrm{Var}(P)| \cdot 2(|\mathrm{Var}(Q)| + 4)$ nodes, which is a contradiction.

Hence, for trees with more than $2 \cdot |\mathrm{Var}(P)| \cdot 2(|\mathrm{Var}(Q)| + 5)$ nodes, one of the two cases must apply. We now show that, in each of the above cases, we can construct a tree t_{small} in $L(P) - L(Q)$ which is smaller than t'.

Case 1. Consider the path from u to v as in the description of Case 1. We now show how we obtain t_{small} from t'. First, we change the label of every node in $\mathit{Nodes}(t') - T$ to a new Σ-symbol # that does not appear in Q. Call the obtained tree $t'_\#$. Notice that such a Σ-symbol always exists because Q only makes use of a finite subset of our infinite labeling alphabet. This clearly preserves satisfaction of P and non-satisfaction of Q. Next, we remove the parent of v from $t'_\#$, by making v a child of its grandparent, thereby obtaining tree t_{small}. We next show that t_{small} is indeed the tree we are looking for.

First of all, notice that t_{small} still models P, as θ is still a satisfaction of P on t_{small}. Furthermore, towards a contradiction, suppose there is a satisfaction θ_Q for Q on t_{small}. As the length of the path ρ' from u to v in t_{small} is at least $|\mathrm{Var}(Q)| + 3$, there is at least one interior node w of ρ' such that w does not have any siblings in t_{small} and such that no variable of Q is assigned to w by θ_Q. Partition $\mathrm{Var}(Q)$ into the set Y of variables assigned by θ_Q to nodes of the subtree of t_{small} rooted at w, and the set X of those that are not. Then Q cannot contain a predicate $\mathit{Child}(x, y)$ for any variables $x \in X$ and $y \in Y$. Now we can insert a node labeled # between w and its child, obtaining a tree isomorphic to $t'_\#$. It is straightforward to verify that θ_Q is a satisfaction for Q on this new tree, and thus also on $t'_\#$. This is a contradiction.

Case 2. Consider the nodes u, v as in the description of Case 2 and consider the nodes between u and v. Without loss of generality, assume that u is to the left of v. We obtain t_{small} from t' similarly as in Case 1. First, just as in Case 1, we change the label of every node in $\mathit{Nodes}(t') - T$ to a new Σ-symbol # that does not appear in Q and we call the obtained tree $t'_\#$. Then, we remove the next sibling of u in $t'_\#$, and we adapt the *NextSibling* relation to reflect this change (that is, we fill in the new next sibling of u). Doing so, we obtain the tree t_{small}. The argument why t_{small} is the tree we are looking for is completely analogous to the argument in Case 1, except that we now need to consider *NextSibling*-axes instead of *Child*-axes. Notice that all nodes between u and v are leaves in t'.

The above two cases can be repeated until we have a witness tree of size at most $2 \cdot |\mathrm{Var}(P)| \cdot 2(|\mathrm{Var}(Q)| + 5)$. □

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 $2 \cdot |\mathrm{Var}(P)| \cdot 2(|\mathrm{Var}(Q)| + 5)$, 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 $\mathit{CQ}(\mathit{Child}, \mathit{NextSibling}, \mathit{NextSibling}^*, \mathit{NextSibling}^+)$, $\mathit{CQ}(\mathit{Child}^*, \mathit{Child}^+)$, and $\mathit{CQ}(\mathit{Following})$ [10], the above algorithm gives us a coNP upper bound for containment for these fragments. We can therefore state the following theorem.

Theorem 7.

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

3.3. coNP *lower bounds*

For the coNP lower bounds, we will reduce from the complement of either the Shortest Common Supersequence problem or the Shortest Common Superstring problem, both of which are known to be NP-complete [14,8]. 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 8. *Containment is* coNP*-hard for $\mathit{CQ}(\mathit{NextSibling}^+)$, $\mathit{CQ}(\mathit{NextSibling}^*)$, $\mathit{CQ}(\mathit{Child}^+)$, $\mathit{CQ}(\mathit{Child}^*)$, and $\mathit{CQ}(\mathit{Following})$.*

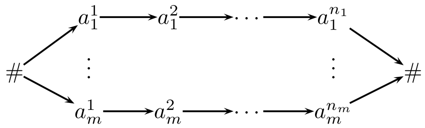


Fig. 9. Structure of query P in the proof of Theorem 8.

Proof. All cases are proved by a reduction from the complement of SHORTEST COMMON SUPERSEQUENCE. To this end, let (S, 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. 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 Fig. 9, 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 along 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 Fig. 9, the arrows in Fig. 9 now denote *NextSibling*-axes, and that all the a_j^i-labeled nodes are connected to a common parent by *Child*-axes.

Theorem 9. *Containment is* coNP-*hard for* $CQ(\mathit{Child}, \mathit{NextSibling})$.

3.4. Π_2^P lower bounds

The Π_2^P lower bounds in this section will all be obtained by a reduction from ∀∃ 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 a standard proof showing that positive 1-in-3 SAT is NP-complete [15].

Lemma 2. ∀∃ *positive* 1-*in*-3 *SAT is* Π_2^P-*complete.*

Proof. Membership of the problem in Π_2^P is trivial. For Π_2^P-hardness, we reduce from ∀∃ 3SAT. First, we convert a ∀∃ 3SAT formula ϕ into a ∀∃ 1-in-3 SAT formula ϕ'. Second, we show how to get rid of negative literals.

Let $C = (x \vee y \vee z)$ be a clause of ϕ (here, x, y, z are literals, not variables). We introduce six new existentially quantified variables, a, b, c, d, e, f, to simulate C. To do this, we introduce the new clauses $(x \vee a \vee d)$, $(y \vee b \vee d)$, $(a \vee b \vee e)$, $(c \vee d \vee f)$, and $(z \vee c)$. It is easy to verify that there is an assignment of truth values to the new variables that makes exactly one literal per clause true if and only if at least one of the literals x, y, z is true.

We show next how to make all literals positive. For each variable x that appears both positively and negatively, we replace all occurrences of $\neg x$ with a new existentially quantified variable $\bar{x}$, and add the clause $(x \vee \bar{x})$. This makes sure that exactly one of x and $\bar{x}$ is assigned true.

Finally, we show how to ensure that each clause contains exactly three literals. Suppose that we have a clause $(x \vee y)$. We introduce four new existentially quantified variables f, a, b, c and rewrite $(x \vee y)$ as $(x \vee y \vee f)$, $(f \vee a \vee b)$, $(f \vee b \vee c)$, and $(a \vee b \vee c)$. The intuition is that f can never be chosen to be true and that, if f is false, we can choose b to be true. □

Theorem 10. *Containment is* Π_2^P-*complete for* $CQ(\mathit{Child}, \mathit{Child}^+)$ *and* $CQ(\mathit{Child}, \mathit{Child}^*)$.

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

The proof is an adaptation of a proof by Gottlob et al., showing that the query complexity of evaluation for $CQ(\mathit{Child}, \mathit{Child}^+)$ is NP-hard [10]. We reduce from ∀∃ positive 1-in-3 SAT, which is Π_2^P-complete (see Lemma 2).

For the readability of this proof, we will first assume that each tree node can carry multiple labels. We explain at the end of the proof how it can be modified to work for the standard definition of labeled trees, where each node has only one label.

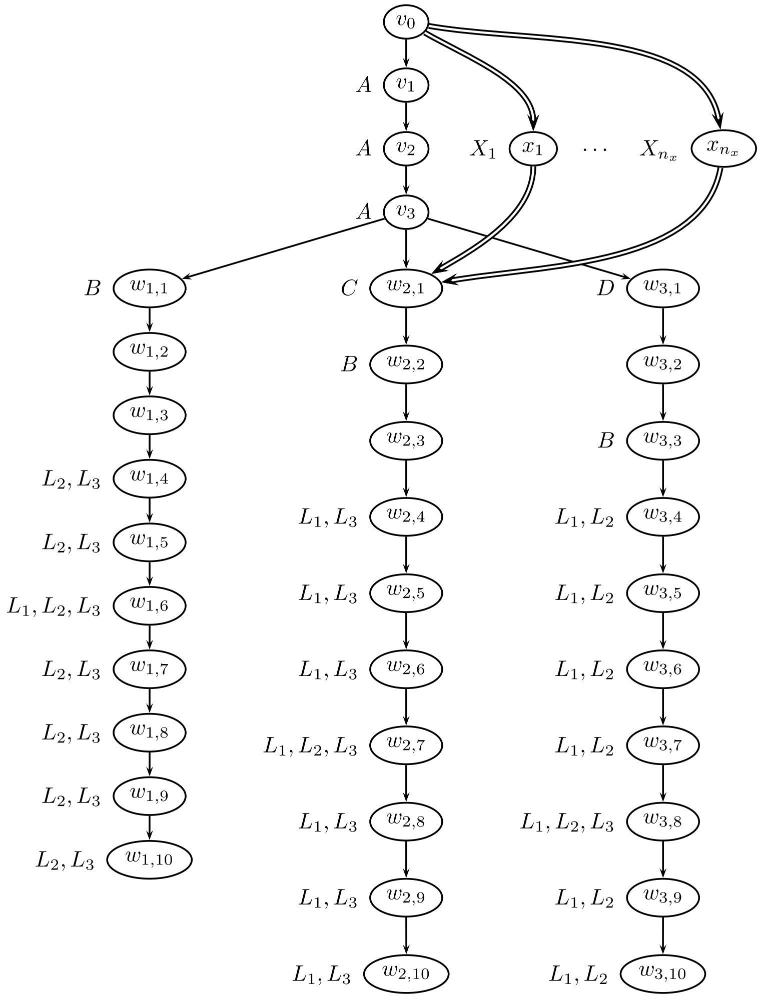


Fig. 10. Illustration of the definition of query P in the proof of Theorem 10.

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

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

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 a $\forall\exists$ 1-in-3 SAT solution for the original formula if and only if there is one for Φ. We need the extension ϕ of our original formula for handling the x-variables in our proof.

Let query P be defined as in Fig. 10, where single lines represent the *Child* axis, double lines represent the Child^+ axis, the symbols inside the nodes are variables of P and the symbols to the left of nodes are their 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 ($1 \leqslant j \leqslant m + n_x$, $i \neq j$, $1 \leqslant k, l \leqslant 3$).

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 $i = m + 1, \ldots, m + n_x$, $X_{i-m}(a_i)$.

Before we show that the reduction is correct, we start with an observation. Consider the set of minimal models of P. It is easy to see that this set is not empty, and every minimal model of P has the shape of the tree in Fig. 10 with the addition that, for every $i = 1, \ldots, n_x$, at least one of the nodes v_1, v_2, v_3 is labeled with X_i. Let T_P be the subset of the minimal models such that, for each X_i, precisely one of v_1, v_2, v_3 is labeled X_i. We refer to T_P as the set of *intended models*.

The following observation is immediate from the monotonicity of conjunctive queries (Observation 2) and the fact that each $t \in L(P)$ has a $\{Child\}$-subtree in T_P.[4]

Observation 11. The following statements are equivalent:

- $\forall t_P \in T_P$: $t_P \models Q$,
- $\forall t \in L(P)$: $t \models Q$.

We show that the reduction from $\forall\exists$ positive 1-in-3 SAT to containment for $CQ(Child, Child^+)$ is correct, that is,

$$\forall \bar{x} \exists \bar{y} C_1, \ldots, C_m \quad \Leftrightarrow \quad L(P) \subseteq L(Q).$$

($\Rightarrow$) Assume that, for every truth assignment σ_x: $\{x_1, \ldots, x_{n_x}\} \to \{\text{true}, \text{false}\}$, there exists a truth assignment $\sigma_y : \{y_1, \ldots, y_{n_y}\} \to \{\text{true}, \text{false}\}$ such that each clause C_i, $1 \leqslant i \leqslant m$, contains precisely one true literal under σ_x and σ_y. We show that $t_P \models Q$ for every $t_P \in T_P$. According to Observation 11, this implies that $L(P) \subseteq L(Q)$.

Let t_P be an arbitrary, but fixed, tree in T_P. Then there exists a satisfaction θ_P of P on t_P. From θ_P, we define a truth assignment $\sigma_{t_P} : \{x_1, \ldots, x_{n_x}\} \to \{\text{true}, \text{false}\}$ as follows:

- if $\theta_P(v_2)$ is labeled X_i, then we set $\sigma_{t_P}(x_i) = \text{true}$;
- otherwise, we set $\sigma_{t_P}(x_i) = \text{false}$.

By definition, σ_{t_P} assigns a truth value to every x_i, $1 \leqslant i \leqslant n_x$. Hence, there exists a $\sigma_y : \{y_1, \ldots, y_{n_y}\} \to \{\text{true}, \text{false}\}$ such that each clause C_i, $1 \leqslant i \leqslant m$, contains precisely one true literal under σ_{t_P} and σ_y. From σ_y, we now construct a truth assignment $\sigma'_y : \{y_1, \ldots, y_{n_y}, y'_1, \ldots, y'_{n_x}, y''_1, \ldots, y''_{n_x}\} \to \{\text{true}, \text{false}\}$ as follows:

- for each $i = 1, \ldots, n_y$, $\sigma'_y(y_i) = \sigma_y(y_i)$;
- if $\theta_P(v_2)$ is labeled X_i, then we set $\sigma'_y(y'_i) = \sigma'_y(y''_i) = \text{false}$;
- otherwise, if $\theta_P(v_1)$ is labeled X_i, then we set $\sigma'_y(y'_i) = \text{true}$ and $\sigma'_y(y''_i) = \text{false}$;
- otherwise, we set $\sigma'_y(y''_i) = \text{true}$ and $\sigma'_y(y'_i) = \text{false}$.

It is easy to see that each clause $C_1, \ldots, C_m$ contains precisely one true literal under σ_{t_P} and σ_y if and only if each clause $C_1, \ldots, C_m, C_{m+1}, \ldots, C_{m+n_x}$ contains precisely one true literal under σ_{t_P} and σ'_y.

We will show how σ_{t_P} and σ'_y induce a satisfaction θ_Q of Q on t_P. Let $\sigma : \{1, \ldots, m + n_x\} \to \{1, 2, 3\}$ be defined as $\sigma(i) = k'$ if and only if the k'-th literal in C_i is true under σ_{t_P} and σ'_y. Notice that σ is total and well defined. We first define a valuation θ_Q of Q on t_P and then show that all query atoms are satisfied. We set

- $\theta_Q(a_i) = \theta_P(v_{\sigma(i)})$ for each $i = 1, \ldots, m + n_x$;
- $\theta_Q(b_i) = \theta_P(w_{\sigma(i),\sigma(i)})$ for each $i = 1, \ldots, m + n_x$; and
- $\theta_Q(c_{k,l,i,j}) = \theta_P(w_{\sigma(i),5+k-l+\sigma(j)})$ for each variable $c_{k,l,i,j}$.

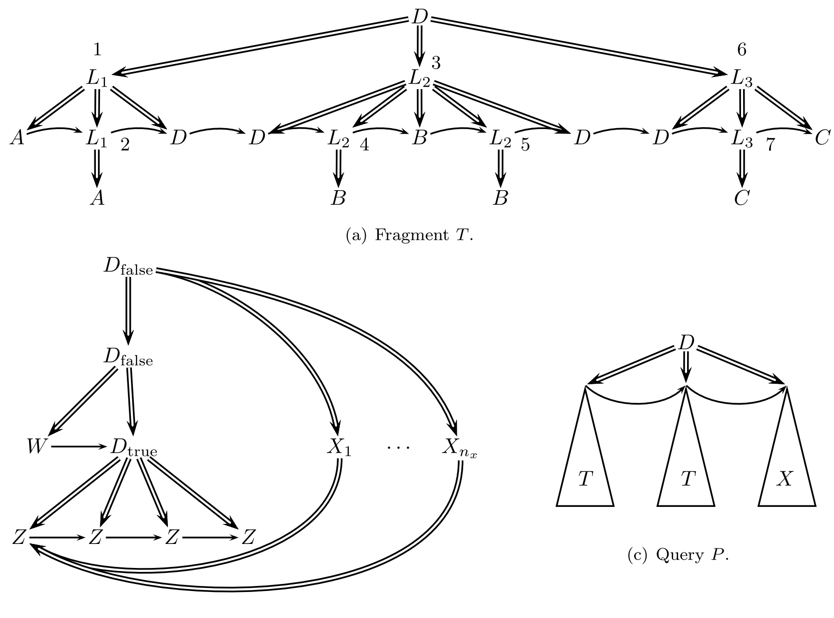
We now prove that θ_Q is a satisfaction of Q on t_P. Our choice of θ_Q implies that the variables a_i and b_i are mapped to nodes with labels A and B, respectively. Furthermore, $\theta_Q(b_i) = \theta_P(w_{\sigma(i),\sigma(i)})$ can be reached from $\theta_Q(a_i) = \theta_P(v_{\sigma(i)})$ with three child-steps. For every variable of the form $c_{k,l,i,j}$, we know that $\theta_Q(c_{k,l,i,j}) = \theta_P(w_{\sigma(i),5+k-l+\sigma(j)})$ is always a descendant of $\theta_P(w_{\sigma(i),\sigma(i)})$. If $\sigma(i) \neq k$, then $\theta_Q(c_{k,l,i,j}) = \theta_P(w_{\sigma(i),5+k-l+\sigma(j)})$ has label L_k because $4 \leqslant 5+k-l+\sigma(j) \leqslant 10$ and the nodes $\theta_P(w_{\sigma(i),4}), \ldots, \theta_P(w_{\sigma(i),10})$ all have (at least) the two labels $L_{k'}$ for which $\sigma(i) \neq k'$. If $\sigma(i) = k$, then $\sigma(j) = l$. By going $8+k-l$ steps downward from $\theta_P(v_{\sigma(j)})$, passing through $\theta_P(w_{k,k})$, we reach node $\theta_P(w_{k,5+k})$, which has label L_k. Since $\theta_Q(c_{k,l,i,j}) = \theta_P(w_{\sigma(i),5+k-l+\sigma(j)}) = \theta_P(w_{k,5+k})$, the query atoms $Child^{8+k-l}(a_j, c_{k,l,i,j})$ are satisfied. For each $i = m+1, \ldots, m+n_x$, we have that $\sigma(i) = k$ if and only if $\theta_P(v_k)$ is labeled X_{i-m}. Hence, for each $i = m+1, \ldots, m+n_x$, $\theta_Q(a_i) = \theta_P(v_{\sigma(i)})$ is labeled X_{i-m}. Therefore, θ_Q is indeed a satisfaction of Q on t_P and $t_P \models Q$.

($\Leftarrow$) Assume that $t_P \models Q$ for every $t_P \in T_P$. We show that, for each truth assignment $\sigma_x : \{x_1, \ldots, x_{n_x}\} \to \{\text{true}, \text{false}\}$, there exists a truth assignment $\sigma_y : \{y_1, \ldots, y_{n_y}\} \to \{\text{true}, \text{false}\}$ such that each clause C_i, $1 \leqslant i \leqslant m$, contains precisely one true literal under σ_x and σ_y.

Let $\sigma_x : \{x_1, \ldots, x_{n_x}\} \to \{\text{true}, \text{false}\}$ be a truth assignment. We define the tree t_x as the tree implied by the variables and *Child*-axes in Fig. 10 with the additions that, for each $i = 1, \ldots, n_x$,

- if $\sigma_x(x_i) = \text{true}$, then only v_2 is labeled X_i; and
- if $\sigma_x(x_i) = \text{false}$, then only v_1 is labeled X_i.

[4] Recall the definition of $\mathcal{R}$-subtrees from Section 2.



(a) Fragment T.

(b) Fragment X.

(c) Query P.

Fig. 11. Definition of query P in the proof of Theorem 13.

Obviously, t_x is in T_P and therefore t_x models P. Hence, $t_x \models Q$.

Let θ be a satisfaction of Q on t_x. We show that θ induces a truth assignment $\sigma_y : \{y_1, \ldots, y_{n_y}\} \to \{\text{true}, \text{false}\}$ such that each clause C_i, $1 \leqslant i \leqslant m$ contains precisely one true literal under σ_x and σ_y. We first show that θ induces a truth assignment $\sigma'_y : \{y_1, \ldots, y_{n_y}, y'_1, \ldots, y'_{n_x}, y''_1, \ldots, y''_{n_x}\} \to \{\text{true}, \text{false}\}$ such that each clause C_i, $1 \leqslant i \leqslant m + n_x$ contains precisely one true literal under σ_x and σ'_y.

To this end, if $\theta(a_i) = v_k$, we interpret this as the k-th literal of clause C_i being chosen to be true. Obviously, under any valuation of Q on t_x, we select precisely one literal from each clause C_i in this way. Because of the constructions of t_x, we know that the literal x_i is selected for clause C_{m+i} if and only if $\sigma_x(x_i) = \text{true}$. We have to verify that if a literal L occurs in two clauses C_i and C_j and we select L in C_i, we also select L in C_j. Let L be the k-th literal of C_i and the l-th literal of C_j, and let $\theta(a_i) = v_k$ (i.e., L is selected in C_i). Then $\theta(c_{k,l,i,j}) = w_{k,5+k}$ because that is the only node below $\theta(b_i) = w_{k,k}$ that has label L_k. The query contains the atom $Child^{8+k-l}(a_j, c_{k,l,i,j})$ for variable $c_{k,l,i,j}$. From node $w_{k,5+k}$, by $8 + k - l$ upward steps we arrive at node v_l. Hence $\theta(a_j) = v_l$, and we select L from clause C_j.

The truth assignment σ_y we are looking for is σ'_y restricted to $\{y_1, \ldots, y_{n_y}\}$.

To conclude the proof, we discuss how to deal with the multiple node labels. The idea is to replace each variable z of P that has k labels by $k+1$ variables $\{z_0, z_1, \ldots, z_k\}$. In the construction from Fig. 10, z_0 takes the place of z, while each z_i, $1 \leqslant i \leqslant k$ carries one of the k labels, and is required to be a child of z_0 ($Child(z_0, z_i)$). In query Q, the same transformation is then used.

Finally, we describe what changes have to be made for the proof to work in the $CQ(Child, Child^*)$ case. In the reduction, we replace each pair of atoms $Child^+(v_0, X_i), Child^+(X_i, w_{2,1})$ of P (for $1 \leqslant i \leqslant n_x$) with the pair $Child^*(v_1, X_i)$, $Child^*(X_i, v_3)$. In Q, we can simply replace $Child^+$ with $Child^*$. The correctness proof is then analogous. □

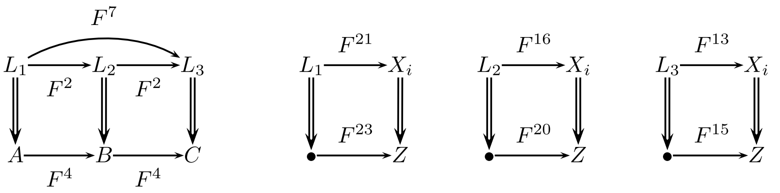
Theorem 12. *Containment is Π_2^P-hard for $CQ(Child, Following)$.*

Proof. We adapt the proof of Theorem 10 by simulating $Child^+$ with $Child$ and $Following$. To this end, we begin by equipping each of the variables u in query P defined in Fig. 10 that has an outgoing $Child^+$-axes by two "dummy" children z_1 and z_2. These new variables are used nowhere else, and get a new Σ-label # that doesn't appear in the queries P and Q. Now, whenever $Child^+(u, v)$ is used in one of the queries, we can replace it by

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

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



(a) The 1-in-3 gadget.

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

Fig. 12. Gadgets for the definition of query Q in the proof of Theorem 13.

Table 3
The function NAND(k,l) [10].

$k \backslash l$	1	2	3
1	10	13	18
2	5	8	13
3	2	5	10

Proof. We first explain the proof for $CQ(Child^+, Following)$ and argue later that it works analogously for $CQ(Child^*, Following)$. Let $\forall\bar{x}\exists\bar{y}C_1,\ldots,C_m$ be an instance of $\forall\exists$ positive 1-in-3 SAT. Let $\bar{x} = \{x_1,\ldots,x_{n_x}\}$ and let $\bar{y} = \{y_1,\ldots,y_{n_y}\}$. We can assume that no clause contains a particular literal more than once.

We construct two queries, P and Q, over the labeling alphabet $\{A, B, C, D, D_{\text{true}}, D_{\text{false}}, L_1, L_2, L_3, X_1,\ldots,X_{n_x}, W, Z\}$ such that $L(P) \subseteq L(Q)$ if and only if $\forall\bar{x}\exists\bar{y}C_1,\ldots,C_m$ 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 (Theorem 5.2 in [10]).

The construction of query P is illustrated in Fig. 11. Here, every double-lined edge represents a *Child*$^+$-axis and every directed edge represents a *Following*-axis. Fig. 12 depicts the gadgets from which query Q will be constructed. For improved readability, we adopt the terminology of the proof by Gottlob 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 Fig. 12(a) by v_1, v_2, and v_3, respectively. Moreover, we annotate the query fragment T in Fig. 11(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 Fig. 11(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 14. (See [10].) 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.

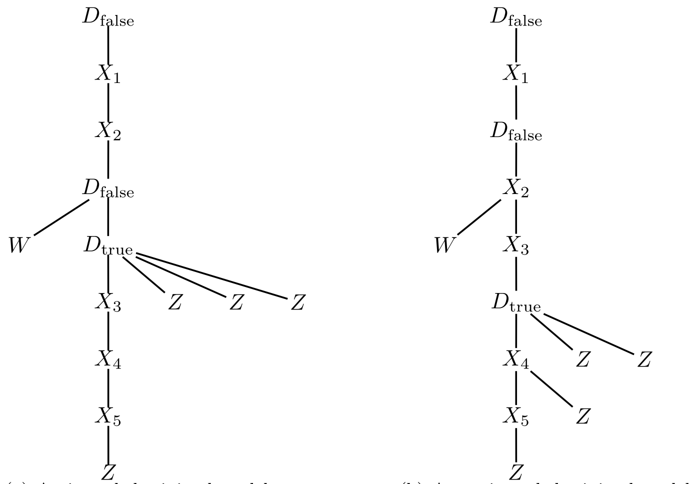
Given a clause C, we interpret a satisfaction θ in which variable v_k is mapped to its topmost position as the selection of the k-th literal from C to be true. Hence, the 1-in-3 gadget would ensure that, on $t_{\min}$, exactly one variable of clause C is selected to be true.

We now define the query P as in Fig. 11(c). That is, P contains two copies of the fragment T, followed by a copy of the X-fragment from Fig. 11(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 the 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 $\bar{x}$ and $\bar{y}$ actually make one literal per clause of $\forall\bar{x}\exists\bar{y}C_1,\ldots,C_m$ 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 $\bar{x}$-variables. Roughly, we interpret x_i as "true" if X_i can be reached from the W-labeled node with a *Following*-step, and as "false" otherwise (see Fig. 11(b)). For example, all X_i-labeled descendants of the D_{true} node are interpreted as "true", and all X_i-labeled ancestors of the lower D_{false} node are interpreted as "false".

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 Fig. 12(b).

Formally, query Q is defined as follows. Each clause C_i is represented by two copies of the 1-in-3 gadget of Fig. 12(a), a left copy Q_i and a right copy Q_i'. The two sets of subqueries $Q_1,\ldots,Q_m, Q_1',\ldots,Q_m'$ are connected as follows. Consider the function NAND(k,l) in Table 3, as defined by Gottlob et al. In a left and right copy of the tree $t_{\min}$ that would match the left and right copy of T in P, we can enforce that two variables, x and y, labeled L_k and L_l in their respective subqueries in Q, cannot both match the topmost node labeled L_k, respectively L_l, in the left, respective right, copy of $t_{\min}$ by adding an atom of the form $Following^{\text{NAND}(k,l)}(x, y)$ to the query Q. To see this, we exemplify the case where $k = l = 1$. Observe that, from



(a) An intended minimal model. (b) A non-intended minimal model.

Fig. 13. Minimal models of the X-fragment of P.

the top L_1 node in the left copy of T in Q, one can reach the upper L_1 node in the right copy of T with 9 *Following*-steps, but not with 10. The lower L_1 node, however, can be reached with 10 *Following*-steps. Hence, $\text{NAND}(1,1) = 10$. The other cases are analogous.

So, for each pair of clauses C_i, C_j, variable $x \in \text{Var}(Q)$ such that Q_i contains the atom $L_k(x)$, and variable $y \in \text{Var}(Q)$ such that Q'_j contains the atom $L_l(y)$, if

- the k-th literal of C_i also occurs in C_j and
- the k-th literal of C_i and the l-th literal of C_j are different,

then we add an atom $\textit{Following}^{\text{NAND}(k,l)}(x, y)$ to Q. As in the proof by Gottlob et al., these query atoms make sure that if a literal is chosen to be true in one clause, it is chosen to be true in other clauses as well; and that both copies Q_i and Q'_i of the query gadget of each clause make the same choice of selected literal.

Finally, we need to make sure that the assignment to the universally quantified variables from $\bar{x}$ defined by a minimal model of P is respected. If Q_i contains the unary atom $L_k(x)$ and the k-th literal of C_i is a (universally quantified) variable x_l from $\bar{x}$, then we add a copy of the gadget $\text{varX}(k,l)$ to the query, in which we identify the L_k-labeled node with the query variable x.

Intuitively, the gadget $\text{varX}(k,l)$ ensures that if x_l is the k-th literal of Q_i, then Q_i picks the value for x_l that is generated by the tree. We explain this more formally. First, observe that, if t_P is a minimal model of P, then the label X_l occurs precisely once in t_P. (Because, if X_l occurs multiple times, t_P is not minimal.) Next, we need to define our *intended minimal models*. A minimal model t_P of P is an intended minimal model if

1. the D_{true}-labeled node is a child of the lower D_{false}-labeled node;
2. the W-labeled node is a child of the lower D_{false}-labeled node; and
3. the three rightmost Z-labeled nodes are children of the D_{true}-labeled node.

Fig. 13 contains an intended (left) and a non-intended minimal model (right) of the X-fragment of P.

Let t_P be an intended minimal model of P. We say that *t_P picks x_l to be true* if the X_l-labeled node can be reached with a *Following*-step from the W-labeled node in t_P (i.e., if it is a descendant of D_{true}), and we say that t_P picks x_l to be false otherwise (i.e., if it is an ancestor of the lower D_{false}-labeled node). We can now make the following observation:

Observation 15. Let t_P be an intended minimal model of P. Then, for every satisfaction θ of Q on t_P, the following holds. If the k-th literal of C_i is a universally quantified variable x_l, then θ selects the k-th literal x_l of C_i to be true on t_P if and only if t_P picks x_l to be true.

Proof. Observation 15 can be easily verified by testing the possible homomorphisms from the X-variable gadgets of Q (Fig. 12(b)) to the query P (Fig. 11). We provide a proof for one of the cases, the arguments for all the other cases are analogous. For the direction from left to right, say that θ chooses the first literal x_l of C_i to be true on t_P. Then θ also matches the L_1-labeled node of the leftmost X-variable gadget in Fig. 12(b) to the upper L_1-labeled node in the first subtree of t_P. From here, the W-labeled node in t_P can be reached by 20 *Following* steps, but not by 21. This means that θ

must match the X_l-labeled node as a descendant of D_{true}. Also note that the upper L_1-labeled node in the first subtree of t_P has a descendant (the left A-labeled child) from which the Z-descendant of X_l can be reached with 23 *Following* steps.

For the direction from right to left, say that θ chooses the first literal x_l of C_i to be false on t_P. Then θ also matches the L_1-labeled node of the leftmost X-variable gadget in Fig. 12(b) to the lower L_1-labeled node in the first subtree of t_P. From here, the W-labeled node in t_P can be reached by 21 *Following* steps, so in principle we can still match X_l everywhere. However, the X-variable gadget also requires that the L_1-labeled node has a descendant (which can only be its A-labeled child in t_P), from which we can reach a Z-labeled descendant of X_l with 23 *Following* steps. This is only possible if the X_l-labeled node occurs as an ancestor of D_{true}, which means that t_P chooses x_l to be false. This concludes the proof of Observation 15. □

This concludes the reduction for Theorem 13. We proceed to proving that the reduction is also correct. That is, we show that

$$\forall\bar{x}\exists\bar{y}C_1,\dots,C_m \quad \Leftrightarrow \quad L(P)\subseteq L(Q).$$

($\Leftarrow$) Suppose that $L(P)\subseteq L(Q)$. We show that, for each truth assignment $\sigma_x:\{x_1,\dots,x_{n_x}\}\to\{\text{true},\text{false}\}$, there exists a truth assignment $\sigma_y:\{y_1,\dots,y_{n_y}\}\to\{\text{true},\text{false}\}$ such that each clause C_i, $1\leqslant i\leqslant m$ contains precisely one true literal under σ_x and σ_y.

Let $\sigma_x:\{x_1,\dots,x_{n_x}\}\to\{\text{true},\text{false}\}$ be an arbitrary but fixed truth assignment. We define an intended minimal model t_x of P as follows. The root of t_x has three children. The first and second child correspond to the left and right T-subquery of P, respectively. For the two copies of subquery T in P, t_x has a child relation for every occurrence of a descendant relation in T, and the ordering of the nodes in t_x is given by the *Following*-relations in P. For the fragment X, it is slightly more complicated. The third subtree of t_x has $8+n_x$ nodes, corresponding to the D_{true}, two D_{false}, the W, the four Z, and the n_x X_i-labeled nodes in Fig. 11(b). The structure of the subtree is given by conditions (1)–(3) of an intended minimal model. Because of condition (1), this leaves two possibilities for the X_i-labeled nodes to occur: an X_i-labeled node either occurs between the two D_{false}-labeled nodes (we call this area X^{false}), or it can occur as a descendant of the D_{true}-labeled node (we call this area X^{true}).

The correspondence between σ_x and the third subtree of t_x is now encoded as follows:

- if $\sigma_x(x_i)=\text{true}$, then the label X_i occurs in X^{true} and not in X^{false}; and
- if $\sigma_x(x_i)=\text{false}$, then the label X_i occurs in X^{false} and not in X^{true}.

Obviously, t_x is an intended minimal model of P. As we assumed that $L(P)\subseteq L(Q)$, we have that $t_x\models Q$.

Let θ be a satisfaction of Q on t_x. We show that θ induces a truth assignment $\sigma_y:\{y_1,\dots,y_{n_y}\}\to\{\text{true},\text{false}\}$ such that each clause C_i, $1\leqslant i\leqslant m$ contains precisely one true literal under σ_x and σ_y.

To this end, if x is an L_k-labeled variable of Q_i, the k-th literal of C_i is existentially quantified, and $\theta(x)$ is the topmost position of L_k in t_x, we interpret this as the k-th literal of clause C_i being chosen to be true by σ_y. We argue that σ_y is indeed the truth assignment we are looking for.

As argued in the construction of Q, in every valuation of Q on t_x, the 1-in-3 gadgets select precisely one literal from each clause C_i in this way. Furthermore, the *Following*$^{\text{NAND}(k,l)}$ atoms ensure that if a literal L occurs in two clauses C_i and C_j and we select L in C_i, then we also select L in C_j. Finally, the varX(k,l) gadgets ensure that θ picks the same values for the $x_i\in\bar{x}$ as t_x (Observation 15), and therefore also σ_x. Hence, the existence of θ implies the existence of a valuation σ_y such that each clause C_i, $1\leqslant i\leqslant m$ contains precisely one true literal under σ_x and σ_y.

($\Rightarrow$) Assume that, for every truth assignment $\sigma_x:\{x_1,\dots,x_{n_x}\}\to\{\text{true},\text{false}\}$, there exists a truth assignment $\sigma_y:\{y_1,\dots,y_{n_y}\}\to\{\text{true},\text{false}\}$ such that each clause C_i, $1\leqslant i\leqslant m$ contains precisely one true literal under σ_x and σ_y.

We show that $L(P)\subseteq L(Q)$. To this end, let T_P be the set of minimal models of P, including non-intended minimal models. Analogously as in the proof of Theorem 10, we make the following observation.

Observation 16. The following are equivalent:

- $\forall t_P\in T_P$: $t_P\models Q$,
- $\forall t\in L(P)$: $t\models Q$.

The observation follows from Observation 2, as each tree in $L(P)$ has a $\{\mathit{Child}^+,\mathit{Following}\}$-subtree in T_P.

We show that $t_P\models Q$ for every $t_P\in T_P$. According to Observation 16, this implies that $L(P)\subseteq L(Q)$.

Given $t_P\in T_P$, we define a truth assignment $\sigma_{t_P}:\{x_1,\dots,x_{n_x}\}\to\{\text{true},\text{false}\}$ as follows:

- if the X_i-labeled node can be reached from the W-labeled node by a *Following*-step in t_P, then we set $\sigma_{t_P}(x_i)=\text{true}$;
- otherwise, we set $\sigma_{t_P}(x_i)=\text{false}$.

By definition of P, σ_{t_P} assigns a truth value to every x_i, $1\leqslant i\leqslant n_x$. Hence, there exists a $\sigma_y:\{y_1,\dots,y_{n_y}\}\to\{\text{true},\text{false}\}$ such that each clause C_i, $1\leqslant i\leqslant m$, contains precisely one true literal under σ_{t_P} and σ_y.

We will show how σ_{t_P} and σ_y induce a satisfaction θ of Q on t_P. Let $\tau : \{1, \dots, m\} \to \{1, 2, 3\}$ be defined as $\tau(i) = k$ if and only if the k-th literal in C_i is true under σ_{t_P} and σ_y. Notice that τ is total and well defined. We first define a valuation θ of Q on t_P and then show that all query atoms are satisfied. Let Q_i be a 1-in-3 gadget of Q and let v_1, v_2, and v_3 be the nodes labeled L_1, L_2, and L_3 in Q_i, respectively. By 1–7 we denote the nodes in t_P that correspond to the nodes 1–7 in the left copy of T in P. We set

- $\theta(v_1) = 1$, $\theta(v_2) = 4$, $\theta(v_3) = 7$ if $\tau(i) = 1$;
- $\theta(v_1) = 2$, $\theta(v_2) = 3$, $\theta(v_3) = 7$ if $\tau(i) = 2$; and
- $\theta(v_1) = 2$, $\theta(v_2) = 5$, $\theta(v_3) = 6$ if $\tau(i) = 3$.

By definition of Q_i, θ can be extended to a valuation of Q_i on t_P for each i. We define the valuation of Q_i' on the second subtree of t_P completely analogously. By definition, the *Following*$^{\mathrm{NAND}(k,l)}$ atoms connecting Q_i and Q_i' are also satisfied. It only remains to show that the gadgets varX(k,l) can be satisfied.

We argue that these gadgets can be satisfied by matching each X_l-labeled node in varX(k,l) onto the unique occurrence of X_l in t_P. Thereto, let z_1, z_2, z_3, z_4 be the nodes in t_P that correspond to the four Z-labeled nodes in fragment X, from left to right. If X_l is reachable from the W-node with a *Following*-step, then $\sigma_{t_P}(x_l) = \text{true}$. This means that θ also selects x_l to be true. To satisfy the X-variable gadgets, we can now always map the Z-labeled node in the gadgets to z_1 which is always a descendant of X_l (see also Fig. 13). If X_l is not reachable from the W-node with a *Following* step a descendant of D_{true}, then we can always map the Z-labeled node in the gadgets to z_4 (see also Fig. 13).

This concludes the proof for $CQ(Child^+, Following)$. The proof for $CQ(Child^*, Following)$ is completely analogous. The reason is that, for each occurrence of $Child^+(x, y)$ in P, either x and y bear different alphabet labels, or x has a descendant z with a different alphabet label, from which y can be reached with a *Following*-axis. Hence, y can never be matched to the same node as x. □

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

Corollary 1. *Containment is Π_2^P-hard for $CQ(Child^*, NextSibling^+)$.*

Theorem 17. *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 proof of Theorem 13 can be adapted by the same methods as in the article by Gottlob et al. [10]. 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 18. *Containment is Π_2^P-hard for $CQ(Following, NextSibling)$.*

Proof. Unfortunately, the arguments we use in Theorem 17 do not work seamlessly for *Following* and *NextSibling*$^\alpha$, where $\alpha \in \{1, +, *\}$. Even though we can express that, e.g., y must be a descendant of x by the formula

$$NextSibling(x_1, x) \wedge NextSibling(x, x_2) \wedge Following(x_1, y) \wedge Following(y, x_2)$$

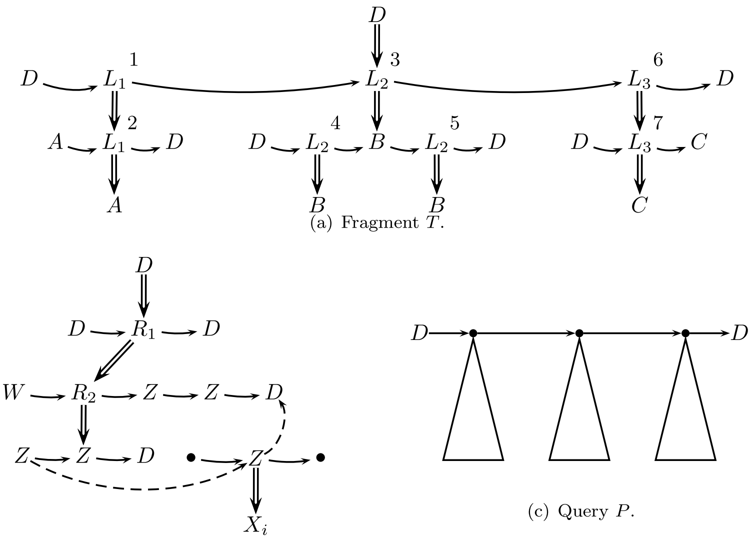
and by giving x and y different labels, the extra introduced nodes x_1 and x_2 for this encoding introduce difficulties for the X_i-labeled nodes of the P-query in the proof of Theorem 17. We therefore need to take a slightly different approach.

Figs. 14 and 15 illustrate how to change P and Q in the proof of Theorem 13. Here, every solid arrow denotes a *NextSibling* axis, every dotted arrow denotes a *Following* axis, and every double line from x to y (where x is above and y is below) denotes the gadget

$$Descendant(x, y) = NextSibling(x_1, x) \wedge NextSibling(x, x_2) \wedge Following(x_1, y) \wedge Following(y, x_2),$$

where x_1 and x_2 are the variables left and right from x in Fig. 14, respectively. It is easy to see that $Descendant(x, y)$ expresses that $Child^+(x, y)$ must hold in all cases: either x and y are labeled differently, or one of their siblings is labeled differently.

Furthermore, the placement of the X_i-labeled nodes is different from their placement for Theorem 13. Here, the idea is that the X_i labeled nodes are either descendants of the Z-labeled descendant of the R_2-labeled node, or descendants of one of the two rightmost Z-labeled right siblings of the R_2-labeled node.

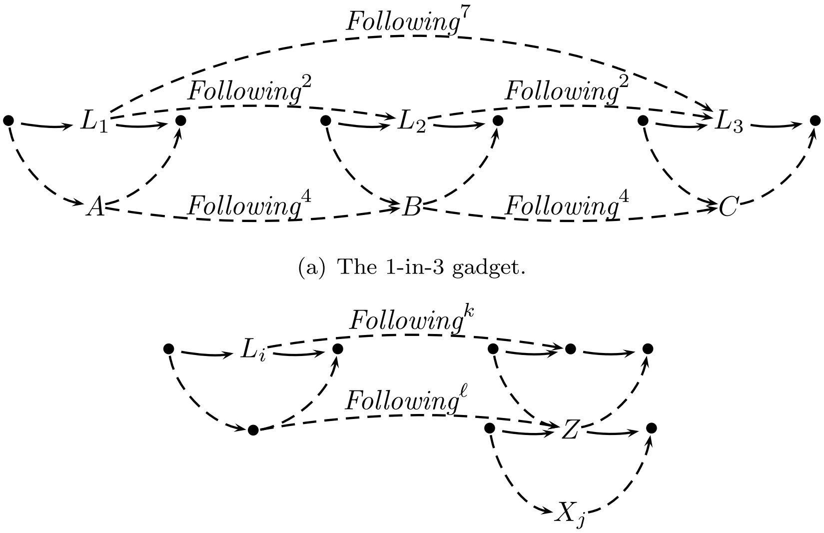


(a) Fragment T.

(b) Fragment X.

(c) Query P.

Fig. 14. Definition of query P in the proof of Theorem 18.



(a) The 1-in-3 gadget.

(b) New X-gadgets. For varX(i, j) the values of k and ℓ are 25 and 28 for $i = 1$, 20 and 25 for $i = 2$, and 17 and 20 for $i = 3$, respectively.

Fig. 15. How to adapt Q for Theorem 18.

Given that a double line denotes the above *Descendant* gadget, the 1-in-3 gadget of Q are almost the same as in Fig. 12(a). The only difference is that the L_1, L_2, and L_3 labeled nodes need extra left and right siblings in the gadget to express the descendant relation. This is illustrated in Fig. 15(a). The gadgets Q_i and Q_i' are then wired in precisely the same manner as in the proof of Theorem 13.

The most significant change in the Q-query is in the varX(i, j)-gadgets for the X-variables. How to adapt these gadgets is illustrated in Fig. 15(b). The remainder of the proof is analogous to the proof of Theorem 13. □

Theorem 19. *Containment is Π_2^P-hard for CQ(Following, NextSibling$^+$) and CQ(Following, NextSibling*).*

Proof. The reduction for *CQ*(*Following*, *NextSibling*$^+$) is analogous to the one in Theorem 18, i.e., we can replace every *NextSibling* in the proof of Theorem 18 with a *NextSibling*$^+$. The reduction of Theorem 18 can be adapted to a reduction for *CQ*(*Following*, *NextSibling**) by replacing every *NextSibling*(x, y) in P with *NextSibling*$^*(x, xy) \wedge H(xy) \wedge$ *NextSibling*$^*(xy, y)$, where H is a new label; replacing every *NextSibling*(x, y) in Q with *NextSibling*$^*(x, y)$; and by changing the number of *Following*-steps in the X-gadgets of Q.

Consider a varX(i, j)-gadget from the proof of Theorem 18, depicted in Fig. 15(b). We observe that in an intended minimal model of P, if we take k minimal following steps from a node labeled L_i, we will actually be following a *NextSibling*-axes $k - 4 + i$ times. Thus, after the modification of P, where each *NextSibling*-axes has been doubled, we will need $2(k - 6 + i) + 6 - i = 2k - 6 + i$ *Following* steps to reach the corresponding node. For l, the corresponding new number is $2k - 5 + i$.

This means, that in the new varX(i, j)-gadgets (see Fig. 15(b)), when $i = 1$ we get $k = 45$ and $l = 52$. For $i = 2$ we get $k = 26$ and $l = 47$, while the numbers for $i = 3$ are $k = 31$ and $l = 38$.

The correctness proofs for both cases are obtained through Observation 1. □

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 1. *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 a tree that

- contains the set T of nodes of t onto which variables are matched by θ;
- contains, for each nonempty $S \subseteq T$, the least common ancestor of the nodes in S;
- 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 valuation for Q on t', and verify in polynomial time that the valuation is actually a satisfaction, i.e., that all atoms of Q are satisfied. □

4.1. PTIME upper bounds

Theorem 20. *Satisfiability is in PTIME for CQ(Child) and CQ(NextSibling).*

Proof. First, we apply the chase to Q and obtain query chase(Q) (cf. Section 2.4). It should be clear that chase(Q) is satisfiable if and only if Q is satisfiable.

Our new query chase(Q) is satisfiable if and only if the following two conditions are met.

1. There is no variable x in chase(Q) that has two labels, i.e., there is no x such that both $a(x)$ and $b(x)$ are atoms of chase(Q), with $a \neq b$.
2. There are no cycles in chase(Q), i.e., the query graph of chase(Q) is acyclic.

Each of these conditions can be tested in polynomial time. □

Before we state the next theorem, we introduce the concept of a siblinghood, which will be useful in our next two proofs.

Definition 4. In a tree t, a *siblinghood* is a subset S of Nodes(t) such that all nodes in S have the same parent, i.e., there is a node $u \in Nodes(t)$ such that $Child(u, v)$ holds for all $v \in S$.

Theorem 21. *Satisfiability is in PTIME for CQ(NextSibling$^+$, NextSibling*, Following) and CQ(Child$^+$, Child*).*

Proof. We start by checking for cycles. If the query graph of Q has a cycle on which at least one edge is labeled by *NextSibling*$^+$, *Following*, or *Child*$^+$, then Q is unsatisfiable. Unlike in the proof of Theorem 20, however, a query may have cycles of *Child** (resp., *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 $a \neq b$. If there is, Q is unsatisfiable. Allowed cycles, i.e., those consisting of only *Child** (resp. *NextSibling**) axes and without multiple labels, can be removed by identifying all variables on the cycle. In the remainder of the proof, we assume that the query is cycle free.

For *CQ*(*NextSibling*$^+$, *NextSibling**, *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 that belong to the same *siblinghood*. As a first step, we note that if Q is satisfiable, then Q', obtained by replacing all *NextSibling**-atoms of Q by *NextSibling*$^+$-atoms is also satisfiable.

Indeed, if θ is a satisfaction of Q on tree t, $NextSibling^*(x, y)$ 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 ordering on the variables. A topological sorting according to this partial ordering gives us a string of variables such that if $NextSibling^+(x, y)$ or $Following(x, y)$ 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. This shows that any $Q \in CQ(NextSibling^+, NextSibling^*, Following)$ that passes the acyclicity tests at the beginning of this proof is satisfiable.

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

Theorem 22. *Satisfiability is in* PTIME *f orCQ*$(Child, NextSibling)$ *and* $CQ(Child, NextSibling^+, NextSibling^*)$.

Proof. For a conjunctive query Q in either of the classes above, we let Q_{ns} be the subquery obtained by removing all *Child*-atoms from Q. Similarly, let Q_{c} be the subquery obtained by removing all $NextSibling^{\alpha}$-atoms, for $\alpha \in \{1, +, *\}$. We note that if variables x and y belong to the same maximal connected component of Q_{ns}, then, for any tree $t \in L(Q)$, any satisfaction for Q on t has to assign x and y to nodes that belong to the same siblinghood of t.

We first present an algorithm for checking satisfiability of queries Q in $CQ(Child, NextSibling)$. If the query graph of Q has cycles, it is always unsatisfiable. Thus we assume that Q is acyclic.

In the description of the algorithm, we make use of a copy P of Q, which will be modified by the algorithm. Actually we can see P as being defined on equivalence classes $[x]$ of variables, which the algorithm sometimes merges. At the beginning, P thus has one singleton class $[x]$, for each variable $x \in \mathrm{Var}(Q)$.

The algorithm first iterates over the following three steps, and stops when no merges occurred in the last iteration.

1. For each pair $[x], [y]$, check whether there exist $[z], [z']$ that belong to the same connected component of P_{ns} and such that $Child([x], [z])$ and $Child([y], [z'])$ are atoms of P. If this is the case, try to merge $[x]$ and $[y]$. This try fails if there are $x \in [x]$ and $y \in [y]$ such that $a(x)$ and $b(y)$ are atoms of Q, for $a \neq b$. If the check fails, P is unsatisfiable.
2. For each maximal connected component of P_{ns}, check satisfiability as in the proof of Theorem 20. When this procedure merges classes of variables, carry these merges over to P_{ns}, P_{c}, and P.
3. For each maximal connected component of P_{c}, check satisfiability as in the proof of Theorem 20. When this procedure merges classes of variables, carry these merges over to P_{ns}, P_{c}, and P.

If the iteration stops without reporting unsatisfiability, the algorithm performs one extra test. This is an acyclicity test on an extended query graph G_P^+ of P, namely the graph where *Child*-edges are, as usual, considered directed, while *NextSibling*-edges are considered undirected (or, equivalently, can be traversed in both directions). In G_P^+, we test whether there is a cycle that uses at least one *Child*-edge. If this is the case, P is unsatisfiable. Indeed, if t is a tree and θ a valuation for P on t such that $\theta([y])$ is a child of $\theta([x])$, then $\theta([x])$ can never be reached from $\theta([y])$ by taking any number of *Child*- or *NextSibling*-steps in t.

Notice that the steps of the iteration above only try to merge variables that always have to be assigned to the same tree node by every satisfaction for P. They report unsatisfiability if such a merge fails or if a merge has introduced cycles. This immediately implies that Q is unsatisfiable as well.

If all tests above succeed, we claim that P is satisfiable.

(a) Since step (1) of the iteration cannot merge any more classes, we know that for each connected component C of P_{ns}, there is *at most one* variable $[x]$ of P that can have child axes to variables in C.
(b) Since step (2) cannot merge any more classes, we know that each connected component of P_{ns} is string-shaped, that is, forms a non-branching sequence of variable classes, connected by *NextSibling*-axes.
(c) Since step (3) cannot merge any more classes, we know that P_{c} is forest-shaped.
(d) We also know that no two variable classes that belong to the same connected component of P_{ns} are connected via *Child*-axes.

Let $C = \{C_1, \ldots, C_k\}$ be the maximal connected components of P_{ns}. We define the relation $\prec$ on $C \times C$ by $C_i \prec C_j$ if there are variables $[x] \in \mathrm{Var}(C_i)$ and $[y] \in \mathrm{Var}(C_j)$ such that $Child([x], [y])$ is an atom of P. We argue that the directed graph $G_\prec = (C, \prec)$ of $\prec$ is a forest. To do this, we must show that $G_\prec$ has no cycles, and that there are no $i \neq j \neq k$ such that $C_i \prec C_k$ and $C_j \prec C_k$.

A cycle in $G_\prec$ would immediately imply a cycle in G_P^+ containing at least one child axis, which the algorithm has already tested for. Thus $G_\prec$ is acyclic.

Suppose there are $i \neq j \neq k$ such that $C_i \prec C_k$ and $C_j \prec C_k$. Then there must be $[x] \in \mathrm{Var}(C_i)$, $[y] \in \mathrm{Var}(C_j)$, and $[z], [z'] \in \mathrm{Var}(C_k)$ such that both $Child([x], [z])$ and $Child([y], [z'])$ are atoms of P. This is ruled out by (a) above, and thus a contradiction.

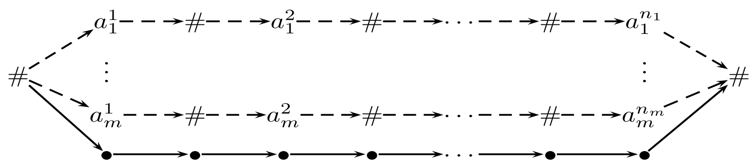


Fig. 16. Gadget for the proof of Theorem 23.

Given this knowledge, we can construct a witness tree t and an accompanying satisfaction θ_P as follows. For each maximal connected component C_i of P_{ns}, we construct a siblinghood S_i modeling the component, and let θ_P assign variables to nodes in the straightforward way. This is always possible by (b).

For each pair of variables $[x], [y]$ such that *Child*$([x],[y])$ is an atom of P, we add a child edge from $\theta_P([x])$ to each node in the siblinghood $\theta_P([y])$ belongs to. Since we know that $G_{\prec}$ is a forest, the resulting structure is a forest. To complete the construction, we add a new root node, and connect it to the root of each tree in the forest. It immediately follows that all *Child*- and *NextSibling*-atoms are satisfied. Thus θ_P is a satisfaction of P on t. It is straightforward to see that θ_Q, defined by $\theta_Q(x) = \theta_P([x])$ is a satisfaction of Q on t.

For $CQ(Child, NextSibling^+, NextSibling^*)$, the process is similar. The differences lie in steps (1) and (2) of the iteration. In (1), we allow merging variables that are connected with the *NextSibling**-axes, but not those connected by the *NextSibling*$^+$-axes. In (2), satisfiability checking for P_{ns} is done as in the proof of Theorem 21. This means that after the iteration terminates, it is not necessarily the case that each connected component of P_{ns} is string-shaped. Each such component is, however, satisfiable, and we can, as argued in the proof of Theorem 21, find a string model for it by considering a topological sorting. □

4.2. NP lower bounds

Theorem 23. *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) $CQ(Child, Following)$.

Proof. All reductions are from Shortest Common Supersequence. For cases (1)–(5), the reductions are very similar. Let S and k be an instance of SHORTEST COMMON SUPERSEQUENCE. For each of the fragments, we define a conjunctive query P such that P is satisfiable if and only if there exists a shortest common supersequence for S of length at most k. 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.

The construction of P is depicted in Fig. 16. The dotted arrows denote *Child*$^+$, *Child**, *NextSibling*$^+$, *NextSibling**, or *Following*-axes and the solid arrows denote *Child* or *NextSibling* axes, whichever are relevant for the fragment under consideration. The bulleted ("•") nodes represent unlabeled variables. The idea is that the path with the solid arrows contains $2k-1$ bulleted nodes. Hence, there exists a tree model for the query if and only if there exists a shortest common supersequence for S of length at most k.

For fragment (6), the above reduction does not work. It can be fixed, however, by using the same trick as in Theorem 12, i.e., replacing all occurrences of $Child^+(u,v)$ in the proof for $CQ(Child, Child^+)$ by

$$Child(u,z_1) \wedge Child(u,z_2) \wedge Following(z_1,v) \wedge Following(v,z_2). \qquad \square$$

5. 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. In a recent paper [4] we studied the containment, satisfiability, and validity problems for conjunctive queries with respect to schemas. It turns out that the presence of a schema dramatically increases the complexity. In particular containment of CQs with respect to DTDs is shown to be 2EXPTIME-complete.

References

[1] M. Arenas, L. Libkin, XML data exchange: Consistency and query answering, J. ACM 55 (2) (2008).
[2] A. Berglund, S. Boag, D. Chamberlin, M.F. Fernández, M. Kay, J. Robie, J. Siméon, XML Path Language (XPath) 2.0, Technical report, World Wide Web Consortium, January 2007, http://www.w3.org/TR/xpath20/.
[3] H. Björklund, W. Martens, T. Schwentick, Conjunctive query containment over trees, in: 11th International Symposium on Database Programming Languages (DBPL), 2007, pp. 66–80.
[4] H. Björklund, W. Martens, T. Schwentick, Optimizing conjunctive queries over trees using schema information, in: 33rd International Symposium on Mathematical Foundations of Computer Science (MFCS), 2008, pp. 132–143.
[5] S. Boag, D. Chamberlin, M.F. Fernández, D. Florescu, J. Robie, J. Siméon Xquery, 1.0: An XML query language, Technical report, World Wide Web Consortium, January 2007, http://www.w3.org/TR/xquery/.
[6] E.P.F. Chan, R. van der Meyden, Containment and optimization of object-preserving conjunctive queries, SIAM J. Comput. 29 (4) (2000) 1371–1400.
[7] J. Clark, S. DeRose, XML Path Language (XPath) version 1.0, Technical report, World Wide Web Consortium, 1999, http://www.w3.org/TR/xpath/.
[8] J. Gallant, D. Maier, J.A. Storer, On finding minimal length superstrings, J. Comput. System Sci. 20 (1) (1980) 50–58.
[9] G. Gottlob, C. Koch, R. Pichler, L. Segoufin, The complexity of XPath query evaluation and XML typing, J. ACM 52 (2) (2005) 284–335.
[10] G. Gottlob, C. Koch, K.U. Schulz, Conjunctive queries over trees, J. ACM 53 (2) (2006) 238–272.
[11] J. Hidders, Satisfiability of XPath expressions, in: 9th International Workshop on Database Programming Languages (DBPL), 2003, pp. 21–36.
[12] P.G. Kolaitis, M.Y. Vardi, Conjunctive query containment and constraint satisfaction, J. Comput. System Sci. 61 (2) (2000) 302–332.
[13] G. Miklau, D. Suciu, Containment and equivalence for a fragment of XPath, J. ACM 51 (1) (2004) 2–45.
[14] K.J. Räihä, E. Ukkonen, The shortest common supersequence problem over binary alphabet is NP-complete, Theoret. Comput. Sci. 16 (2) (1981) 187–198.
[15] T. Schaefer, The complexity of satisfiability problems, in: 10th Annual ACM Symposium on Theory of Computing (STOC), 1978, pp. 216–226.
[16] B. ten Cate, C. Lutz, The complexity of query containment in expressive fragments of XPath 2.0, in: 26th International Symposium on Principles of Database Systems (PODS), 2007, pp. 73–82.
[17] S. Vansummeren, On deciding well-definedness for query languages on trees, J. ACM 54 (4) (2007).