Optimizing Conjunctive Queries over Trees using Schema Information*

Henrik Björklund, Wim Martens**, and Thomas Schwentick

Technical University of Dortmund

Abstract. We study the containment, satisfiability, and validity problems for conjunctive queries over trees with respect to a schema. We show that conjunctive query containment and validity are 2EXPTIME-complete w.r.t. a schema (DTD or Relax NG). Furthermore, we show that satisfiability for conjunctive queries w.r.t. a schema can be decided in NP. The problem is NP-hard already for queries using only one kind of axis. Finally, we consider conjunctive queries that can test for equalities and inequalities of data values. Here, satisfiability and validity are decidable, but containment is undecidable, even without schema information. On the other hand, containment w.r.t. a schema becomes decidable again if the "larger" query is not allowed to use both equalities and inequalities.

1 Introduction

In the context of relational databases, select-project-join queries are the ones most commonly used in practice. These queries are also known in database theory as *conjunctive queries*. The *containment problem* for conjunctive queries P and Q asks whether Q returns (at least) all answers of P. Ever since the seminal paper of Chandra and Merlin [5], conjunctive query containment has been a pivotal research topic; it is the most intensely researched form of query optimization in database theory. Moreover, the conjunctive query containment problem is essentially the same as the conjunctive query evaluation problem [5], and the Constraint Satisfaction Problem (CSP) in Artificial Intelligence [13].

The more recent rise of semi-structured data and XML initiated the investigation of conjunctive queries over trees [11]. As in the relational case, conjunctive queries over trees provide a very clean and natural querying formalism. XPath and (non-recursive) XQuery queries can both be naturally translated into conjunctive queries. However, as pointed out by Gottlob et al. [11], their applications are not at all limited to XML; they are also used for Web information extraction, as queries in computational linguistics, dominance constraints, and in higher-order unification.

As a matter of fact, containment for queries on tree-structured data was previously mainly studied for fragments of XPath 1.0. The investigations therefore concentrated on *acyclic* conjunctive queries (see, e.g., [16, 17]).

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

** Supported by a grant from the Nordrhein Westfälische Akademie der Wissenschaften.

In contrast to the relational setting, for conjunctive queries over trees, evaluation is not the same problem as containment. In relational databases, containment $P \subseteq Q$ holds if an only if there is a homomorphism from the canonical database of Q to the canonical database of P. Over trees, the existence of such a homomorphism is a sufficient, but not a necessary condition for containment [2].

Conjunctive query containment over trees is therefore investigated directly in [2], but was also treated more implicitly in the form of XPath 2.0 static analysis in, e.g., [12, 14, 21]. We elaborate on the relation with these papers below. The results in [2] were encouraging, as the complexities (compared with acyclic queries) did not increase too much: they remained inside Π_2^P.

The present paper extends our previous work [2] in the sense that we now take schema information into account, and that we consider queries that can test for equality and inequality of data values. In this framework, we study the complexities of the validity, satisfiability, and containment problems. Whereas our previous work outlined a quite complete picture of conjunctive query containment without schemas, one has to admit that, in practice, schema information is highly relevant. In XML, schema information is available for most documents, and the chances of being able to optimize queries are much better when it is taken into account. On the other hand, as we will see in this paper, there is also a tradeoff: the complexity of conjunctive query containment over trees is much higher with schema information than without.

Our work can be summarized as follows. First, we study conjunctive queries that cannot compare data values. Our main technical result here is that the practically most relevant problem, conjunctive query containment w.r.t. a DTD, is already 2EXPTIME-hard for queries using only the *Child* and *Child*$^+$ axes.[1] This result is quite surprising when one compares it to the known results for XPath 1.0 containment. For XPath 1.0, adding DTD information to the problem usually "only" increases the complexity from coNP [16] to (at most) EXPTIME [17, 15]. Here, however, the complexity immediately jumps from Π_2^P to 2EXPTIME when DTDs are taken into consideration. In particular, the problem can provably not be solved in polynomial space in general. On the other hand, it remains in 2EXPTIME even when conjunctive queries can use all axes and the much more expressive Relax NG schemas are considered. In contrast, the satisfiability problem for even the most general conjunctive queries w.r.t. Relax NG schemas is in NP. Unfortunately the satisfiability problem is also already NP-hard for very simple cases using only DTD information.

Finally, we turn to the containment problem for queries that can compare data values for equality ($\sim$) and inequality ($\not\sim$). When data values are involved, static analysis problems are generally known to become undecidable very quickly. We show that conjunctive query containment is no exception: already without schema information, it is undecidable. However, the good news is that even very slight restrictions of this most general case become decidable, even without increasing the complexity over the setting without data values.

[1] Actually, we show hardness already for the validity problem.

Boolean versus n-ary queries The conjunctive queries in our paper are *boolean* queries, i.e., they evaluate either to *true* or *false* on a tree. Our complexity results also carry over to containment for conjunctive queries that return an n-ary relation when evaluated on a tree.

Related work We discuss the relation of our paper to some of the above mentioned work. Most relevant to us are the papers by ten Cate and Lutz [21], by David [8] (which evolved independently from ours), and by Lakshmanan et al. [14]. The connection with Hidders' work [12] is explained more elaborately in [2]. Hidders considers XPath 2.0 satisfiability, but does not take schema information into account. Ten Cate and Lutz study query containment for expressive fragments of XPath 2.0, which is closely related to our conjunctive queries. They also take schema information into account (at least for DTDs and XML Schema Definitions) and get 2EXPTIME-completeness, but their queries have negation, disjunction, and union while conjunctive queries do not.

The precise relation between our conjunctive queries and XPath 2.0 is not entirely obvious. Conjunctive queries are at least as expressive as the XPath 2.0 fragment that consists of Core XPath 1.0 without union, disjunction or negation, but augmented with the XPath 2.0 path intersection operator (see [21]). This implies that our upper bound proofs also apply to this XPath 2.0 fragment. On the other hand, such XPath expressions are syntactically constrained and cannot use path intersection arbitrarily. Our lower bound proofs can, however, also be adapted to these XPath 2.0 expressions. In this light, our results significantly strengthen the lower bound proof of Theorem 20 in [21] when DTD information is considered, since we do not make use of negation or disjunction.[2]

David studies the complexity of satisfiability for Boolean combinations of *data tree patterns* with respect to DTDs [8]. Different fragments are investigated, and the complexity results range from NP to undecidable. This formalism is on the surface quite similar to CQs with data value predicates, but there are some decisive differences. First, the data tree patterns are always tree-shaped, like XPath queries without path intersection. Second, the semantics used in [8] is injective, i.e., two variables cannot be assigned the same node, unlike the one for CQs. This means that boolean combinations of data tree patterns are in general more expressive but exponentially less succinct than CQs.

Lakshmanan et al. study satisfiability, with and without schema information, of tree pattern queries, where the tree patterns are also equipped with a node identity operator and can compare data values. In particular, they claim (Theorem 3.2 in [14]) that query satisfiability for queries with structural constraints, Value Based Constraints (VBCs) and no wildcards is in PTIME. However, it is NP-complete.[3] The results of the paper do not really overlap with our results on satisfiability, since they only consider a limited, non-recursive, form of DTDs.

[2] Without DTD information, ten Cate and Lutz still have 2EXPTIME-completeness due to the presence of negation, but conjunctive query containment is Π_2^P-complete.

[3] Here, structural constraints include node identities and VBCs allow comparison of data values to constants. One of our NP-hardness proofs can be easily adapted to this case. However, we do not conclude PTIME = NP.

Furthermore, there is a large amount of work on static analysis for XPath 1.0 (see, e.g., [1, 10, 15–17, 22]). XPath 1.0 relates to our conjunctive queries in a similar way as XPath 2.0, except that XPath 1.0 does not have a path intersection operator. In other words, complexity lower bounds for XPath 1.0 sometimes carry over to conjunctive queries. We indicate this in the paper whenever relevant.

Due to space constraints, most proofs have been omitted and will appear in the full version of the paper.

2 Preliminaries

2.1 Trees

By Σ we always denote a finite alphabet. The trees we consider are rooted, ordered, finite, labeled, unranked trees, which are directed from the root downwards. That is, we consider finite trees in which nodes can have arbitrarily many children, which are ordered from left to right. We view a tree t as a relational structure over a finite number of unary labeling relations $a(\cdot)$, for $a \in \Sigma$, and binary relations $\mathit{Child}(\cdot,\cdot)$ and $\mathit{NextSibling}(\cdot,\cdot)$. Here, $a(u)$ expresses that u is a node with label a, and $\mathit{Child}(u,v)$ (respectively, $\mathit{NextSibling}(u,v)$) expresses that v is a child (respectively, the right sibling) of u.

The reason that we can restrict ourselves to a finite set of labels is that an XML schema defines the set of labels allowed in a tree. In the rare cases where we consider trees without schema information, we also consider the set of possible labels to be infinite.

In addition to *Child* and *NextSibling*, we use their transitive closures (denoted Child^+ and $\mathit{NextSibling}^+$) and their transitive and reflexive closures (denoted Child^* and $\mathit{NextSibling}^*$). We also use the *Following*-relation, which is inspired by XPath [6] and defined as

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

We refer to the binary relations above as *axes*. We denote the set of nodes of a tree t by $\mathrm{Nodes}(t)$. For a node u, we denote by $\mathrm{lab}^t(u)$ the unique a such that $a(u)$ holds in t. We often omit t from this notation when t is clear from the context. By $\mathrm{root}(t)$ we denote the root node of t.

2.2 Conjunctive Queries

Let $X = \{x, y, z, \dots\}$ be a set of variables. A *conjunctive query* (CQ) over alphabet Σ is a positive existential first-order formula without disjunction over a finite set of unary predicates $a(x)$ where each $a \in \Sigma$, and the binary predicates *Child*, Child^+, Child^*, *NextSibling*, $\mathit{NextSibling}^+$, $\mathit{NextSibling}^*$, and *Following*. In this paper, we will mainly focus on Boolean satisfaction of conjunctive queries. We will therefore consider conjunctive queries without free variables, and we also consider the constants *true* and *false* to be CQs. As our conjunctive queries do not contain free variables, we sometimes omit the existential quantifiers to

simplify notation. For a conjunctive query Q, we denote the set of variables appearing in Q by $\mathrm{Var}(Q)$. We use $\mathrm{CQ}(R_1, \ldots, R_k)$ or $\mathrm{CQ}(\mathcal{R})$ (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 from Gottlob et al. [11]. That is, let Q be a CQ, and t a tree. A *valuation* of Q on t is a total function $\theta : \mathrm{Var}(Q) \to \mathrm{Nodes}(t)$. A valuation is a *satisfaction* if it satisfies the query, that is, if every atom of Q is satisfied by the assignment. A tree t *models* Q ($t \models Q$) if there is a satisfaction of Q on t. The language $L(Q)$ of Q is the set of all trees that model Q.[4] We denote the complement of $L(Q)$ by $\overline{L(Q)}$.

We sometimes refer to a query as confluent. Intuitively, this means that the atoms of the query, interpreted as directed edges, merge at some point, i.e., the graph they form is not a directed forest. More formally, query Q is confluent if there are three distinct variables $x, y, z \in \mathrm{Var}(Q)$ and binary predicates R_1 and R_2 such that $R_1(x, z)$ and $R_2(y, z)$ are both atoms of Q.

2.3 Schemas

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

Definition 1. A *Document Type Definition (DTD)* over Σ is a triple $D = (\mathrm{Alpha}(D), \mathrm{Rules}(D), \mathrm{start}(D))$ where $\mathrm{Alpha}(D) = \Sigma$, $\mathrm{start}(D) \in \Sigma$ is the start symbol and $\mathrm{Rules}(D)$ is a set of rules of the form $a \to R$, where $a \in \Sigma$ and R is a regular expression over Σ. Here, no two rules have the same left-hand-side.

A tree t *satisfies* D if *(i)* $\mathrm{lab}^t(\mathrm{root}(t)) = \mathrm{start}(D)$ and, *(ii)* for every $u \in \mathrm{Nodes}(t)$ with label a and n children $u_1, \ldots, u_n$ from left to right, there is a rule $a \to R$ in $\mathrm{Rules}(D)$ such that $\mathrm{lab}^t(u_1) \cdots \mathrm{lab}^t(u_n) \in L(R)$. By $L(D)$ we denote the set of trees satisfying D.

We abstract from Relax NG schemas [7] by unranked tree automata, which are formally defined as follows:

Definition 2. A *nondeterministic (unranked) tree automaton (NTA)* over Σ is a quadruple $A = (\mathrm{States}(A), \mathrm{Alpha}(A), \mathrm{Rules}(A), \mathrm{Final}(A))$, where $\mathrm{Alpha}(A) = \Sigma$, $\mathrm{States}(A)$ is a finite set of states, $\mathrm{Final}(A) \subseteq \mathrm{States}(A)$ is the set of final states, and $\mathrm{Rules}(A)$ is a set of transition rules of the form $(q, a) \to L$, where $q \in \mathrm{States}(A)$, $a \in \mathrm{Alpha}(A)$, and L is a regular string language over $\mathrm{States}(A)$.

For simplicity, we denote the regular languages L in A's rules by regular expressions. For our complexity results, it doesn't matter whether the languages L are represented by regular expressions or nondeterministic string automata.

A *run* of A on a tree t is a labeling $r : \mathrm{Nodes}(t) \to \mathrm{States}(A)$ such that, for every $u \in \mathrm{Nodes}(t)$ with label a and children $u_1, \ldots, u_n$ from left to right,

[4] Notice that, as stated in the introduction, we assume that trees only take labels from a finite alphabet Σ. Hence, for a conjunctive query Q, $L(Q)$ also consists of trees over alphabet Σ. In the rare cases where we consider trees without schema information, we state this explicitly.

there exists a rule $(q, a) \to L$ such that $r(u) = q$ and $r(u_1) \cdots r(u_n) \in L$. Note that, when u has no children, the criterion reduces to $\varepsilon \in L$, where ε denotes the empty string. A run is *accepting* if the root is labeled with an accepting state, that is, $r(\text{root}(t)) \in \text{Final}(A)$. A tree t is accepted if there is an accepting run of A on t. The set of all accepted trees is denoted by $L(A)$ and is called a *regular tree language*. We denote the complement of $L(A)$ by $\overline{L(A)}$. In the remainder of the paper, we sometimes view the run r of an NTA on t as a tree over $\text{States}(A)$, obtained from t by relabeling each node u with the state $r(u)$.

From now on, we use the word "schema" to refer to DTDs or NTAs.

2.4 Our Problems of Interest

Definition 3.
- Containment w.r.t. a schema: Given two CQs P and Q, and a schema S, is $L(P) \cap L(S) \subseteq L(Q)$?
- Validity w.r.t. a schema: Given a CQ Q and a schema S, is $L(S) \subseteq L(Q)$?
- Satisfiability w.r.t. a schema: Given CQ Q and schema S, is $L(Q) \cap L(S) \neq \emptyset$?

All of the above problems are in a sense instances of the containment problem. That is, validity of Q is testing whether $L(\text{true}) \subseteq L(Q)$ w.r.t. S, and satisfiability for Q is testing whether $L(Q) \not\subseteq L(\text{false})$ w.r.t. S.

3 Validity and Containment

3.1 Complexity Upper Bounds

We start the technical part of the paper by settling the upper bound for the containment problem. This is achieved through a standard translation of CQs into NTAs.

Lemma 4. *Let Q be a CQ. There exists an NTA A such that $L(A) = L(Q)$ and A can be computed from Q in exponential time.*

It is now easy to derive the following theorem. We note that Theorem 5 is not new. The 2EXPTIME upper bound is obtained by composing the exponential translation of [11] from CQs to Core XPath and the polynomial time translation of [21] from Core XPath expressions to two-way alternating tree automata. The result now follows as emptiness testing of two-way alternating tree automata is in EXPTIME.

Theorem 5. *Containment of CQs w.r.t. an NTA is in 2EXPTIME.*

3.2 Complexity Lower Bounds

In this section, we prove the following result.

Theorem 6. *Validity of CQ(Child,Child$^+$) w.r.t. a tree automaton is 2EXPTIME-complete.*

The proof of the above theorem is long and rather technical. We sketch it's most interesting parts below.

The upper bound in Theorem 6 follows from Theorem 5. We show the corresponding lower bound by reduction from the word problem for alternating exponential space bounded Turing machines, which is 2EXPTIME-hard [4].

An *alternating Turing machine (ATM)* [4] is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0)$ where $Q = Q_\forall \uplus Q_\exists \uplus \{q_a\} \uplus \{q_r\}$ is a finite set of states partitioned into *universal states* from $Q_\forall$, *existential states* from $Q_\exists$, an *accepting state* q_a, and a *rejecting state* q_r. The (finite) input and tape alphabets are Σ and Γ, respectively. We assume that the tape alphabet contains a special *blank* symbol "␣". The *initial state* of M is $q_0 \in Q$. The transition relation δ is a subset of $(Q \times \Gamma) \times (Q \times \Gamma \times \{L, R, S\})$. The letters L, R, and S denote the directions *left*, *right*, and *stay* in which the tape head is moved.

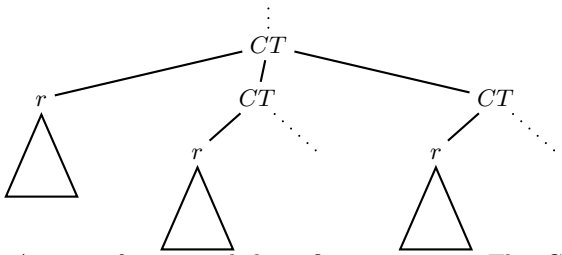
A *computation tree* for an ATM M is a tree labelled by configurations (tape content, reading head position, and internal state) of M such that (1) if node v is labelled by an existential configuration, then v has one child, labelled by one of the possible successor configurations, (2) if v is labelled by a universal configuration, then v has one child for each possible successor configuration, (3) the root is labelled by an initial configuration, and (4) all leaves are labelled by accepting or rejecting configurations. A computation tree is *accepting* if it is finite and all leaves are labelled by accepting configurations.

The overall idea of our proof is as follows. Given ATM M and a word w of length n we construct, in polynomial time, (1) an ATM M_w which accepts the empty word if and only if M accepts w and (2) an NTA A_{CT} that checks most important properties of (suitably encoded) computation trees of M_w, except their consistency w.r.t. the transition relation of M_w. The consistency is tested by the query Q_{CT} that we define. To be precise, Q_{CT} is satisfied by a tree t in $L(A_{CT})$ if and only if the transition relation of M_w is *not* respected by t. This means that Q_{CT} is valid w.r.t. A_{CT}, iff there does not exist a consistent, accepting computation tree for M_w. Since 2EXPTIME is closed under complementation, we conclude that validity of CQs with respect to NTAs is 2EXPTIME-hard.

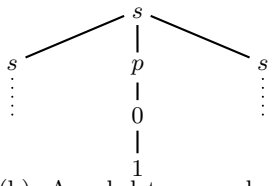
Encoding Computation Trees. The encoding $\mathrm{enc}(t)$ of a possible computation tree t of M_w is illustrated in Fig. 1(a) and obtained from t by replacing each node u of t with a tree t_u, where

- $\mathrm{root}(t_u)$ is labeled CT;
- the leftmost child of $\mathrm{root}(t_u)$ is labeled r (and is the root of the tree encoding the actual configuration at u); and
- for each child u_i of u, $\mathrm{root}(t_u)$ has a subtree $\mathrm{enc}(t/u_i)$ where t/u_i denotes the subtree of t rooted at u_i.

Encoding configurations. The most crucial part of the reduction is to use the query to detect when the transition relation of M_w is violated. To be able to do this, the query must be able to navigate from a node representing tape cell i in



(a) A part of an encoded configuration tree. The CT-labeled nodes define the structure of the actual configuration tree of M_w, while the subtrees with root label r encode the actual configurations of M_w.



(b) A skeleton node which is the left child of its parent, with its two skeleton node children and its navigation gadget.

Fig. 1. The encoded computation tree.

one configuration tree to the node representing cell i in a successor configuration. To this end, we encode configurations as follows.

As we can assume w.l.o.g. that M_w never uses more than 2^n tape cells, we can encode configurations into the leaves of binary trees of height n, where each leaf represents a tape cell. A *configuration tree* is obtained from a full binary tree b of height n as follows. The root gets label r and the other nodes label s. The s-labeled nodes are called *skeleton nodes*. To each skeleton node v we attach a little gadget indicating whether v is a left or a right child in b. More precisely, we attach a path of length 3 labeled with $p, 0, 1$, respectively, to left children and a path labeled with $p, 1, 0$ to right children. Each leaf skeleton node (one that has no skeleton node children) is further provided with the relevant information about the tape cell it represents.

Thus, left and right children can be distinguished by the distance (1 or 2) of their 1-labelled gadget node from their p-labelled gadget node. More precisely, a skeleton node v at level i of a configuration tree and a skeleton node u at level i of a successor configuration tree are both left or both right children, if the nodes v^1 and u^1 with label 1 in their respective gadgets have a common ancestor which has distance $i + 4$ from v^1 and $i + 5$ from u^1.

Comparing configurations. Below, we construct a query $\mathrm{SameCell}(t_1, t_t)$ which is true for two leaf skeleton nodes if and only if they belong to successive configuration trees and represent the same tape cell. In order to do this, we first define successively more complicated subqueries. The first one states that two nodes r_1 and r_2 are roots of two *successive* configuration trees, i.e., configuration trees such that the second encodes a successor configuration of the first.

$$\begin{aligned}\mathrm{Succ}(r_1, r_2) \equiv \exists s_1, s_2 : r(r_1) \wedge r(r_2) \wedge CT(s_1) \wedge CT(s_2) \\ \wedge \mathit{Child}(s_1, r_1) \wedge \mathit{Child}(s_2, r_2) \wedge \mathit{Child}(s_1, s_2)\end{aligned}$$

Next, we define a query $\Phi_i(x, y)$ to state that x and y belong to successive encoded configuration trees and are both at level $i > 0$ of their respective encoded configuration tree. Here, $\mathit{Child}^i(x, y)$ abbreviates the query stating that y can be reached from x by following the *Child*-axis i times.

$$\Phi_i(x, y) \equiv \exists r_1, r_2 : s(x) \wedge s(y) \wedge \mathrm{Succ}(r_1, r_2) \wedge \mathit{Child}^i(r_1, x) \wedge \mathit{Child}^i(r_2, y)$$

Now we can express that x and y fulfil Φ_i and, additionally, that they are either both left children of their parents, or both right children.

$$\begin{aligned}\Psi_i(x, y) \equiv \exists p_x, p_y, t_x, t_y, z : \Phi_i(x, y) \wedge p(p_x) \wedge p(p_y) \wedge 1(t_x) \wedge 1(t_y) \\ \wedge \mathit{Child}(x, p_x) \wedge \mathit{Child}(y, p_y) \wedge \mathit{Child}^+(p_x, t_x) \wedge \mathit{Child}^+(p_y, t_y) \\ \wedge \mathit{Child}^{i+4}(z, t_x) \wedge \mathit{Child}^{i+5}(z, t_y)\end{aligned}$$

Using the above queries, we can now express that s_1 and s_2 are leaf skeleton nodes in successive configuration trees representing the same tape cell. Recall that n is the depth of the encoded configuration trees.

$$\begin{aligned}&\mathrm{SameCell}(s_1, s_2) \equiv \\ &\quad \exists x_1, \ldots, x_{n-1}, y_1, \ldots, y_{n-1} : \bigwedge_{1 \le i < n-1} (\mathit{Child}(x_i, x_{i+1}) \wedge \mathit{Child}(y_i, y_{i+1})) \\ &\qquad \wedge \mathit{Child}(x_{n-1}, s_1) \wedge \mathit{Child}(y_{n-1}, s_2) \wedge \Psi_n(s_1, s_2) \wedge \bigwedge_{1 \le i \le n-1} \Psi_i(x_i, y_i)\end{aligned}$$

DTDs. Actually, the 2EXPTIME lower bound from Theorem 6 can even be strengthened to the case where the schema is just a DTD instead of a tree automaton.

4 Satisfiability

4.1 Complexity Upper Bounds

In this section, we show that testing satisfiability for CQs with respect to a nondeterministic tree automaton is in NP. The idea is a kind of small model property for such queries. We start with the following lemma. The proof is by a standard pumping argument.

Lemma 7. *There is a polynomial p such that if a CQ Q is satisfiable with respect to an NTA A, then there is a tree $t \in L(Q) \cap L(A)$ and a satisfaction θ of Q on t such that for any variables $x, y \in \mathit{Var}(Q)$, the length of the path from $\theta(x)$ to $\theta(y)$ is at most $p(|A|, |Q|)$.*

Lemma 7 gives us the main machinery to prove the general NP upper bound on satisfiability:

Theorem 8. *Satisfiability of CQs with respect to an NTA is in NP.*

4.2 Complexity Lower Bounds

We show that our upper bound for satisfiability w.r.t. a schema is tight, in quite a strong sense. In particular, when considering a DTD as schema, satisfiability is NP-hard for queries using only a single axis, no matter which axis this is.

For the NP lower bounds, we will reduce from the Shortest Common Supersequence problem; or the Shortest Common Superstring problem, both of which are known to be NP-complete [19, 9]. 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 suffix of s.

Theorem 9. *Let Axis be an any element of* $\{$*Child, Child*$^+$*, Child***, NextSibling, NextSibling*$^+$*, NextSibling***, Following*$\}$*. Then Satisfiability of CQ(Axis) w.r.t. a DTD is NP-hard.*

5 Queries with Data Values

A *data tree* is a tree in which each node u, in addition to its label $\mathrm{lab}(u)$, carries a *data value* from a countably infinite data domain Δ (see also [3]).[5] We write $u \sim v$ if two nodes in a data tree have the same data value. Conjunctive queries over data trees can, in addition to the usual predicates, use the binary predicates $\sim$ and $\not\sim$ with the obvious interpretation. We adopt our notation to denote CQ fragments for data values as follows: $\mathrm{CQ}(\sim)$, $\mathrm{CQ}(\not\sim)$, and $\mathrm{CQ}(\sim, \not\sim)$ denote the CQs that use only data equality, only data inequality, and both, respectively, and in which all axes are allowed. For $Q \in \mathrm{CQ}(\sim, \not\sim)$, $L(Q)$ is the set of all *data trees* t such that there exists a satisfaction of Q on t. Schemas do not constrain data values in any way, i.e., the set of data trees $L(A)$ defined by an NTA A is defined precisely as in Section 2.3, but with "tree" replaced by "data tree".

Our problems of interest for queries with data values are the same problems as defined in Section 2.4, but with the new definition of $L(Q)$. We first show that data values do not change the complexity of the satisfiability and validity problems.

Theorem 10. *Satisfiability of CQs*$(\sim, \not\sim)$ *w.r.t. an NTA is NP-complete.*

The proofs of Theorems 8 and 9 straightforwardly carry over to data trees.

The following result follows from Theorem 12, which is strictly stronger.

Theorem 11. *Validity of CQ*$(\sim, \not\sim)$ *w.r.t. an NTA is 2EXPTIME-complete.*

Next, we consider containment w.r.t. a schema. We write $\mathrm{QC}(X|Y)$ for the problem of determining whether $L(P) \cap L(A) \subseteq L(Q)$ for a query $P \in \mathrm{CQ}(X)$,

[5] We assume Δ to contain all the data values we use in our proofs and examples.

$X \setminus Y$	$\sim$	$\not\sim$	$\sim, \not\sim$
$\sim$	2EXPTIME	2EXPTIME	2EXPTIME
$\not\sim$	2EXPTIME	2EXPTIME	undecidable
$\sim, \not\sim$	2EXPTIME	2EXPTIME	undecidable

Table 1. Decidability for QC($X|Y$).

a query $Q \in \mathrm{CQ}(Y)$ and an NTA A. E.g., QC($\sim \mid \sim, \not\sim$) is about containment of queries with data equalities in queries with data equalities and inequalities.

It turns out that the consideration of data values does not change the complexity of the query containment problem for queries P, Q, unless P is allowed to use data inequalities and Q to use equalities and inequalities. In the latter case the problem is undecidable (Theorem 15). We summarize our results for QC($X|Y$) in Table 1.

Theorem 12. *Each of QC($\sim, \not\sim \mid \sim$), QC($\sim, \not\sim \mid \not\sim$), QC($\sim \mid \sim, \not\sim$), w.r.t. an NTA is 2EXPTIME-complete.*

Hence, $\sim$ and $\not\sim$ do no increase the complexity of query containment as long as they do not co-occur in Q. We show next, that the picture changes dramatically if they do co-occur and P uses $\not\sim$.

Theorem 13. *Validity of a disjunction of CQ($\sim, \not\sim$) w.r.t. an NTA is undecidable.*

With a little extra work, Theorem 13 can be extended to the following.

Theorem 14. *QC($\not\sim \mid \sim, \not\sim$) is undecidable.*

Actually, it turns out that if both queries can use $\sim$ and $\not\sim$, the schema automaton from Theorem 14 can be avoided.

Theorem 15. *QC($\sim, \not\sim \mid \sim, \not\sim$) is undecidable, even without a schema.*

6 Conclusion

We studied the query containment and the validity problem for conjunctive queries over trees (1) relative to a schema and (2) taking into account data values. It turned out that in the presence of a schema the complexity of the problem drastically increases. Thus, even though the query language does not have neither negation nor disjunction, it shares the bad complexity (2EXPTIME) of the language in [21].

Not surprisingly, with equalities and inequalities on data values the containment problem even becomes undecidable. Nevertheless, a slight restriction on the occurrence of inequalities yields a decidable problem.

Although conjunctive queries are a very natural query language, future research should identify tractable fragments, in particular with other restrictions

than acyclicity. We found it interesting to observe that, from the lower bound proof of Theorem 6, we can conclude that there does *not* exist an exponential-size tree automaton recognizing the complement language of a conjunctive query.

Corollary 16. *In general, there does not exist an exponential-size nondeterministic tree automaton recognizing* $\overline{L(Q)}$*, where* Q *is a CQ(Child, Child*$^+$*).*

References

1. M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55(2), 2007.
2. H. Björklund, W. Martens, and T. Schwentick. Conjunctive query containment over trees. In *DBPL*, pages 66–80, 2007.
3. M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. In *PODS*, pages 10–19, 2006.
4. A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
5. A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.
6. J. Clark and S. DeRose. XML Path Language (XPath) version 1.0. Technical report, World Wide Web Consortium, 1999. http://www.w3.org/TR/xpath/.
7. J. Clark and M. Murata. Relax NG specification. http://www.relaxng.org/spec-20011203.html, December 2001.
8. C. David. Complexity of data tree patterns over XML documents. In *MFCS*, 2008. To appear.
9. J. Gallant, D. Maier, and J. A. Storer. On finding minimal length superstrings. *JCSS*, 20(1):50–58, 1980.
10. F. Geerts and W. Fan. Satisfiability of XPath queries with sibling axes. In *DBPL*, pages 122–137, 2005.
11. G. Gottlob, C. Koch, and K. U. Schulz. Conjunctive queries over trees. *J. ACM*, 53(2):238–272, 2006.
12. J. Hidders. Satisfiability of XPath expressions. In *DBPL*, pages 21–36, 2003.
13. P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. *JCSS*, 61(2):302–332, 2000.
14. L. V. S. Lakshmanan, G. Ramesh, H. Wang, and Z. Zhao. On testing satisfiability of tree pattern queries. In *VLDB*, pages 120–131, 2004.
15. M. Marx. XPath with conditional axis relations. In *EDBT*, pages 477–494, 2004.
16. G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004.
17. F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *LMCS*, 2(3), 2006.
18. E.L. Post. A variant of a recursively unsolvable problem. *Bull. AMS*, 52(4):264–268, 1946.
19. K.J. Räihä and E. Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *TCS*, 16(2):187–198, 1981.
20. M. Takahashi. Generalizations of regular sets and their application to a study of context-free languages. *Inf. Control*, 27(1):1–36, 1975.
21. B. ten Cate and C. Lutz. The complexity of query containment in expressive fragments of XPath 2.0. In *PODS*, pages 73–82, 2007.
22. P. T. Wood. Containment for XPath fragments under DTD constraints. In *ICDT*, 2003. Full version, obtained through personal communication.

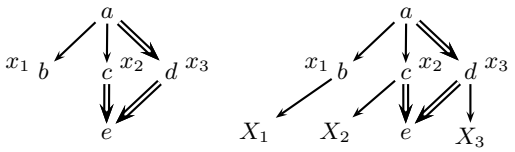


Fig. 2. How to reduce from n-ary queries to 0-ary queries.

A Boolean versus N-ary Queries

In our definition section, we consider conjunctive queries without free variables. This means that we only look at whether a tree models the query or not, and not at the whole set of satisfactions. One can also consider *n-ary conjunctive queries*, i.e., CQs with n free variables, returning a n-ary relation when evaluated on a tree. For two n-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.

First, notice that, for testing whether a query is satisfiable or not, it does not matter whether a query is Boolean or n-ary. So all our results on satisfiability carry over to n-ary queries.

Second, all our other results concern conjunctive queries that can use the *Child*-axis. Using a technique of Miklau and Suciu [16], one can reduce containment for such n-ary queries to containment of Boolean queries. For instance, consider the left query $P(x_1, x_2, x_3)$ in Figure 2. By introducing, for each free variable x_i, a new variable x_i' and adding the atoms $\mathit{Child}(x_i, x_i') \wedge X_i(x_i')$ to the query, where X_i is a new label, the query P', depicted on the right of Figure 2, is obtained. It is now easy to see that, for two queries $P(\overline{x})$ and $Q(\overline{x})$[6] with n free variables, P is contained in Q if and only if $L(P') \subseteq L(Q')$, where P' and Q'. Indeed, the proof is analogous to the one in [16]. For satisfiability, it is of course immediate that the complexities are the same for 0-ary and n-ary queries.

One can generalize this reasoning to incorporate schemas. Such schemas would, e.g., allow the labels X_i as leaf child of every node.

B Conjunctive Queries versus XPath 2.0

Actually, it is technically not difficult to write the queries of our lower bound proofs as XPath 2.0 queries adhering to the grammar

```
locpath ::= '/' locpath | locpath '/' locpath | locstep
            locpath ∩ locpath
locstep ::= axis '::' ntst '[' bexpr ']' ... '[' bexpr ']'
bexpr   ::= bexpr 'and' bexpr | locpath.
axis    ::= 'self' | 'child' | 'parent' | 'descendant' |
            'descendant-or-self' | 'ancestor' |
            'ancestor-or-self' | 'following' |
            'following-sibling' | 'preceding' |
            'preceding-sibling'.
```

where "locpath" is the start production and "ntst" denotes Σ-symbols labeling document nodes or the star '$*$'

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

that matches all tags ("node tests"). All operators come from Core XPath 1.0, except for the *path intersection* operator '$\cap$' which is from XPath 2.0. The semantics of the path intersection operator can be found in [21]. Essentially, a locpath returns a binary relation on a tree, and path intersection returns the intersection of two binary relations.

The most challenging query is the query Q_{CT} from the proof of Theorem 6. Recall that Q_{CT} is graphically presented in Figure 5 in Appendix C.2, which significantly helps for understanding the XPath 2.0 query.

$$Q_{CT} \equiv \bigcap_{i=0}^{n} \Phi_i$$

We define the queries used in Q_{CT}:

$$\Phi_0 = 1/\text{parent}^{n+k+3}::*/\text{child}^{n+k+4}::1$$

For $1 \leq i \leq n$, we define Φ_i as

$$\begin{aligned}\Phi_i = 1/&\text{ancestor}::f/\text{parent}::t/\text{parent}::s/\\ &(\Psi_i^1 \cap \Psi_i^2)/\\ &\text{child}::t/\text{child}::m/\text{descendant}::1\end{aligned}$$

where Ψ_i^1 is defined as

$$\begin{aligned}\Psi_i^1 = ./&\text{child}::p/\text{descendant}::1/\text{parent}^{i+4}::*/\\ &\text{child}^{i+5}::1/\text{ancestor}::p/\text{parent}::s\end{aligned}$$

and Ψ_i^2 is defined as

$$\begin{aligned}\Psi_i^2 = ./&\text{parent}^{i}::r/\text{parent}::CT/\\ &\text{child}::CT/\text{child}::r/\text{child}^{i}::s\end{aligned}$$

The XPath 2.0 version of query Q_{CT} can also be adapted accordingly for the proof of Theorem 19, using predicate expressions.

C Proofs of Section 3

C.1 Complexity Upper Bounds

Proof of Lemma 4: *Let Q be a CQ. There exists an NTA A such that $L(A) = L(Q)$ and A can be computed in exponential time from Q.*

Proof. Essentially, when reading a tree, A guesses the positions where the variables of Q should be placed for a satisfaction of the query and checks whether the correct relations hold between the guessed positions.

As *Child*$^+$, *NextSibling*$^+$, and *Following* can easily be expressed by constant-size formulas only using *Child*, *Child**, *NextSibling*, and *NextSibling**, we only need to consider the latter four axes in this proof.

Intuitively, a state of A is of the form (X_a, X_h, X_d), where X_a, X_h, and X_d are subsets of $\text{Var}(Q)$ such that

– X_a is the set of variables that A guesses to be placed on the *ancestors of the current node*,
– X_h is the set of variables that A guesses to be placed *on the current node*, and
– $X_d \subseteq \mathrm{Var}(Q)$ is the set of variables that A guesses to be placed on *descendants of the current node*.

As A guesses a valuation of Q, we have that a variable of Q can never be placed on a node u and on a descendant of u at the same time. Hence, for each state (X_a, X_h, X_d), the pairwise intersections of X_a, X_h, and X_d are empty.

In order to define A formally, we need to specify $\mathrm{States}(A)$, $\mathrm{Final}(A)$, and $\mathrm{Rules}(A)$. We define these components next:

States(A)**:** The state set of A is the maximal subset of $2^{\mathrm{Var}(Q)} \times 2^{\mathrm{Var}(Q)} \times 2^{\mathrm{Var}(Q)}$ such that the following conditions hold: For each $(X_a, X_h, X_d) \in \mathrm{States}(A)$,

(S1) the pairwise intersections of X_a, X_h, and X_d are empty,
(S2) for each $x, y \in X_h$, Q does not contain atoms of the form $a(x)$ and $b(y)$ with $a \neq b$,
(S3) for each $x \in X_h$ and each $y \in \mathrm{Var}(Q)$ such that $\mathit{Child}(y, x)$ is an atom in Q, $y \in X_a$, and
(S4) for each $x \in X_h$ and each $y \in \mathrm{Var}(Q)$ such that $\mathit{Child}^*(y, x)$ is an atom in Q, $y \in X_h \cup X_a$.

Final(A)**:** A state (X_a, X_h, X_d) of A is in $\mathrm{Final}(A)$ if and only if

(F1) X_a is empty; and
(F2) X_h and X_d partition $\mathrm{Var}(Q)$, i.e., $\mathrm{Var}(Q) = X_h \uplus X_d$.

Rules(A)**:** contains all rules of the form

$$\rho = \big((X_a, X_h, X_d), a\big) \to L, \qquad (\dagger)$$

where

(R1) for each $x \in X_h$, Q does not contain an atom of the form $b(x)$ with $b \neq a$;
(R2) for every string $(X_a^1, X_h^1, X_d^1) \cdots (X_a^n, X_h^n, X_d^n) \in L$, the following holds:
  (a) $X_d = X_h^1 \uplus \cdots \uplus X_h^n \uplus X_d^1 \uplus \cdots \uplus X_d^n$;
  (b) if $x \in X_h$ and Q contains an atom $\mathit{Child}(x, y)$ then there is an $i = 1, \ldots, n$ with $y \in X_h^i$;
  (c) for each $i = 1, \ldots, n$, $X_a^i = X_a \cup X_h$; and
  (d) for each $i = 1, \ldots, n$, if $x \in X_h^i$ and Q contains an atom
    – $\mathit{NextSibling}(x, y)$, then $i < n$ and $y \in X_h^{i+1}$;
    – $\mathit{NextSibling}^*(x, y)$, then there exists a j, $i \leq j \leq n$ such that $y \in X_h^j$.

In order to complete the proof of the lemma, we need to prove that

(1) A can be constructed from Q in exponential time; and
(2) $L(A) = L(Q)$.

(1) It is clear that $\mathrm{States}(A)$ and $\mathrm{Final}(A)$ can be computed in time exponential in $|Q|$. For $\mathrm{Rules}(A)$, we prove that we can compute an non-deterministic finite string automaton (NFA) N that accepts, for every $(X_a, X_h, X_d) \in \mathrm{States}(A)$ and $a \in \mathrm{Alpha}(A)$, the language L in the rule

$$\big((X_a, X_h, X_d), a\big) \to L.$$

As N only reads symbols from $\mathrm{States}(A)$, we don't need to check anymore that (S1)–(S4) hold. Furthermore, (R1) also does not need to be checked by N. This needs to be

checked by the algorithm that constructs A, when deciding whether or not to define a transition rule of the form (†). Hence, we only have to enforce (R2.a)–(R2.d).

We next describe N's accepting condition and the information that N needs to remember when reading a string. As N only needs to maintain a polynomial amount of information at the same time, it should be clear that N needs only an exponentially large set of states. A state of N consists of $(X_h^\cup, X_d^\cup, Y_{ns}, Y_{ns*})$, where the components are defined as follows. When reading a prefix $(X_a^1, X_h^1, X_d^1) \cdots (X_a^k, X_h^k, X_d^k)$ of $(X_a^1, X_h^1, X_d^1) \cdots (X_a^n, X_h^n, X_d^n)$,

- $X_h^\cup := X_h^1 \cup \cdots \cup X_h^k$,
- $X_d^\cup := X_d^1 \cup \cdots \cup X_d^k$,
- $Y_{ns} := \{y \mid x \in X_h^k$ and *NextSibling*(x, y) occurs in $Q\}$,
- $Y_{ns*} := \{y \mid \exists 1 \leq i \leq k, x \in X_h^i,\ y \notin X_h^i \cup \cdots \cup X_h^k$ and *NextSibling*$^*(x, y)$ occurs in $Q\}$.

When reading symbol $(X_a^{k+1}, X_h^{k+1}, X_d^{k+1})$, N checks whether

- $X_h^{k+1} \cap (X_h^\cup \cup X_d^\cup) = \emptyset$, to partially ensure (R2.a);
- $X_d^{k+1} \cap (X_h^\cup \cup X_d^\cup) = \emptyset$, to partially ensure (R2.a);
- $X_a^{k+1} = X_a \cup X_h$, to ensure (R2.c); and
- $Y_{ns} \subseteq X_h^{k+1}$, to ensure (R2.d)'s NextSibling-constraint.

and it changes its state to $(X'^\cup_h, X'^\cup_d, Y'_{ns}, Y'_{ns*})$ as follows:

- $X'^\cup_h = X_h^\cup \cup X_h^{k+1}$;
- $X'^\cup_d = X_d^\cup \cup X_d^{k+1}$;
- $Y'_{ns} = \{y \mid x \in X_h^{k+1}$ and *NextSibling*(x, y) occurs in $Q\}$;
- $Y'_{ns*} = (Y_{ns*} - X_h^{k+1}) \cup \{y \mid x \in X_h^{k+1},\ y \notin X_h^{k+1}$, and *NextSibling*$^*(x, y)$ occurs in $Q\}$.

Finally, N accepts if

- $X_d = X_h^\cup \cup X_d^\cup$, to ensure (R2.a), together with the above conditions on the transitions;
- for each $x \in X_h$ such that *Child*(x, y) occurs in Q, $x \in X_h^\cup$, to ensure (R2.b);
- $Y_{ns} = \emptyset$, to ensure (R2.d)'s NextSibling constraints; and
- $Y_{ns*} \subseteq X_h^k$, to ensure (R2.d)'s NextSibling* constraint.

(2) It can be proved that N recognizes L by a simple induction on the depth of a tree. □

Proof of Theorem 5 *Containment of CQs w.r.t. an NTA is in 2EXPTIME.*

Proof. We reduce the containment problem to testing intersection emptiness of three NTAs, whose sizes are at most doubly exponential in the size of the input. The result then immediately follows, as intersection emptiness testing for three NTAs is in PTIME. Let A be the schema NTA and let P and Q be the queries. According to Lemma 4, we can compute A_P and A_Q such that $L(A_P) = L(P)$ and $L(A_Q) = L(Q)$ in exponential time. It is well-known that the complement NTA $\overline{A_Q}$ of A_Q can be computed in exponential time from A_Q (which is already exponentially large). Hence, the containment problem reduces to testing whether $L(A) \cap L(A_P) \cap L(\overline{A_Q}) = \emptyset$, where each of these three NTAs can be computed in doubly exponential time. □

C.2 Complexity Lower Bounds

We will reduce from the acceptance problem of an exponential space alternating Turing Machine.

Alternating Turing Machines. An *alternating Turing machine (ATM)* [4] is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0)$ where $Q = Q_\forall \uplus Q_\exists \uplus \{q_a\} \uplus \{q_r\}$ is a finite set of states partitioned into *universal states* from $Q_\forall$, *existential states* from $Q_\exists$, an *accepting state* q_a, and a *rejecting state* q_r. The (finite) input and tape alphabets are Σ and Γ, respectively. We assume that the tape alphabet contains a special *blank* symbol "␣". The *initial state* of M is $q_0 \in Q$. The transition relation δ is a subset of $(Q \times \Gamma) \times (Q \times \Gamma \times \{L, R, S\})$. The letters L, R, and S denote the directions *left*, *right*, and *stay* in which the tape head is moved.

A *configuration* of M is a string $w_1 q w_2$ where $w_1, w_2 \in \Gamma^*$ and $q \in Q$. Here, $w_1 q w_2$ denotes that M's work tape contains the word $w_1 w_2$, followed by blanks, that its tape head points to the first symbol of w_2, and that M is in state q. The *successor configurations* of $w_1 q w_2$ are defined analogously as for standard Turing Machines. When $q = q_a$ or $q = q_r$, we say that $w_1 q w_2$ is a *halting configuration*. We can assume w.l.o.g. that each halting configuration has no successor configurations, and that each non-halting configuration has precisely two *different* successor configurations. Furthermore, we can assume w.l.o.g. that each halting configuration is of the form $q w_1 w_2$, i.e., upon halting M moves its tape head entirely to the left.

A *computation tree* of M on an input word w is a (possibly infinite) tree t labeled with configurations of M, such that t's root bears the label $q_0 w$ and, for each node u labeled with $w_1 q w_2$,

- if $q \in Q_\exists$, then u has only one child v and v is labeled with a successor configuration of $w_1 q w_2$,
- if $q \in Q_\forall$, u has two children and, for each successor configuration $w_1' q' w_2'$ of $w_1 q w_2$, u has a child v labeled with $w_1' q' w_2'$, and
- if $q \in \{q_a, q_r\}$, then u is a leaf.

A computation tree is *accepting* if all its branches are finite and each leaf is labeled with a configuration in state q_a. The language $L(M)$ accepted by M is the set of words w for which there exists an accepting computation tree of M on w.

An ATM is said to be *normalized* if each universal step only affects the state of the machine, and additionally, the machine always goes from a universal state to an existential, or vice versa. To be more precise, if $q \in Q_\exists$ (resp., $q \in Q_\forall$) and $a \in \Gamma$, then $\{p \mid ((q, a), (p, b, D)) \in \delta\} \subseteq Q_\forall \cup \{q_a, q_r\}$ (resp., $\subseteq Q_\exists \cup \{q_a, q_r\}$). Moreover, if $q \in Q_\forall$ and $((q, a), (p, b, D)) \in \delta$, then $b = a$ and $D = S$. Any ATM can easily be reduced in polynomial time to a normalized ATM that accepts the same language. Thus, in the sequel, we assume that all ATMs are normalized. There is a (normalized) exponential space bounded ATM whose word problem is 2EXPTIME-hard [4].

In our reduction, we will actually work with an ATM *without input*. In order to do this, given an ATM M whose word problem is 2EXPTIME-complete, and an input word w, we first construct an ATM M_w that, when given the empty word as input, works in space exponential in $|w|$ and accepts if and only if M accepts w. This is achieved by letting M_w start by writing w on its work tape and return to the first tape position. After this, it simulates M.

Let M be a normalized exponentially space bounded ATM and $w \in \Sigma^*$ an input for M of length n. Let $M_w = (Q, \Sigma, \Gamma, \delta, q_0)$ be constructed from M and w as described

above. We may assume that the non-blank portion of the tape of the computation of M_w on the empty word ε is never longer than 2^n.

Goal of the Proof. The goal of our proof is to construct, in polynomial time, a tree automaton A_{CT}, and a conjunctive query Q_{CT} such that Q_{CT} is valid w.r.t. A_{CT} if and only if M_w does not have an accepting computation tree. As 2EXPTIME is closed under complement, this shows that CQ validity w.r.t. an NTA is 2EXPTIME-hard.

Intuitively, A_{CT} will accept trees that encode possible computation trees for M_w. That is, A_{CT} checks whether its input is such an encoded computation tree and that the configurations of M_w are correctly encoded, but will not check consistency w.r.t. the transition relation of M_w, i.e., the relation between successive configurations. This will be taken care of by Q_{CT}. More precisely, Q_{CT} will match in a tree accepted by A_{CT} exactly when a successive configuration does *not* respect the transition relation of M_w. Thus, Q_{CT} is valid w.r.t. A_{CT} if every finite computation tree with correctly encoded configurations contains a violation w.r.t. the transition relation of M_w, i.e., there is no accepting computation tree for M_w.

The Encoding

Encoding Computation Trees. Possible computation trees of M_w will be encoded as illustrated in Figure 1(a). More formally, let t be a computation tree of M_w. The encoded computation tree $\mathrm{enc}(t)$ is obtained from t by replacing each node u with a tree t_u, where

- the $\mathrm{root}(t_u)$ is labeled CT;
- the leftmost child of $\mathrm{root}(t_u)$ is labeled r (and will be the "root" of an encoding of the configuration at u); and
- for each child u_i of u, $\mathrm{root}(t_u)$ has a subtree $\mathrm{enc}(t/u_i)$ where t/u_i denotes the subtree of t rooted at u_i.

Hence, Figure 1(a) shows a fragment of an encoded computation tree representing a universal configuration (left), and its two successor configurations (right). We know that the CT-labeled node on top represents a universal configuration, because it has two CT-labeled children.

Encoding Configurations. We first encode a configuration of M_w as a sequence of 2^n *configuration cells*, and we will discuss later how we encode this sequence of configuration cells as a tree. A *configuration cell* contains the content of a tape cell of M_w, plus some additional information. In particular, the set of configuration cells is partitioned into three types:

- The set BCells of *basic cells* is equal to Γ. These cells represent tape cells that are not currently visited by the tape head, and also weren't visited in the last configuration.
- The set CCells of *current tape-head cells* is equal to $\Gamma \times \delta$. These cells represent tape cells that are currently visited by the tape head. The letter from Γ represents the cell content, while the transition from δ represents the transition by which M_w arrived in the current configuration.

– The set PCells of *previous tape-head cells* is equal to $\Gamma \times (Q \times \Gamma)$. These cells represent tape cells that were visited by the tape head in the last configuration, but aren't in the current one. The letter from Γ represents the current cell content, while the pair from $Q \times \Gamma$ represent the cell content and the machine state in the last configuration.

Let C_1 and C_2 be two sequences of 2^n configuration cells. We will argue how a few simple constraints can ensure that C_2 encodes a valid successor configuration of C_1. To this end, think of C_2 lying on top of C_1 in the obvious manner (i.e., the leftmost configuration cell of C_2 lying on top of the leftmost configuration cell of C_1, etc.). We divide the set of constraints into two: a set of *horizontal* constraints ensuring consistency inside C_1 and inside C_2, and a set of *vertical* constraints ensuring consistency between C_1 and C_2.

The set $H(M_w)$ of horizontal constraints enforce the following rules:

(H1) The only cells allowed to the *left* of a cell $(a, ((q_1, b), (q_2, c, R))) \in$ CCells are cells $(c, (q_1, b)) \in$ PCells.
(H2) The only cells allowed to the *right* of a cell $(a, ((q_1, b), (q_2, c, L))) \in$ CCells are cells $(c, (q_1, b)) \in$ PCells.
(H3) The only cell allowed to the *right* of cell $\textvisiblespace \in \Gamma$ is $\textvisiblespace$.

The set $V(M_w)$ of vertical constraints enforce the following rules.

(V1) On top of a cell $a \in$ BCells, the only allowed cells are a itself and any $(a, ((q_1, b), (q_2, c, m))) \in$ CCells such that $m \in \{L, R\}$. That is, M_w just moved to the current position from the left or from the right. The current position is not overwritten.
(V2) On top of a cell $(a, ((q_1, b), (q_2, c, m))) \in$ CCells, the only allowed cells are
  – any $(d, (q_2, a)) \in$ PCells and
  – any $(d, ((q_2, a), (q_3, d, m'))) \in$ CCells, if $m' = S$.
(V3) On top of a cell $(a, (q, b)) \in$ PCells, the only allowed cells are
  – $a \in$ BCells, and
  – any $(a, ((q_1, b), (q_2, c, m))) \in$ CCells such that $m \in \{L, R\}$.

Figure 3 shows an example of a valid transition from C_1 to C_2 w.r.t. the horizontal and vertical constraints.

The following is now easy to see:

Observation 17. If C_1 is a valid encoding of a configuration, then C_2 is a valid encoding of a successor configuration of C_1 if and only if all conditions (H1–H3, V1–V3) are fulfilled.

Encoding Configurations as Trees. We now describe how the configurations of M_w will be encoded as trees, thereby filling in the remaining structure of the empty r-rooted trees in Figure 1(a).

Notice that there are a polynomial number of possible configuration cells for M_w, say k. In this section, we associate a fixed number i in $\{1, \ldots, k\}$ to each configuration cell, and we refer to it as configuration cell i.

Each sequence of 2^n configuration cells will be represented by a full binary tree of height n, in which each leaf represents a configuration cell. For technical reasons, the configuration cells will not be represented by labels, but rather by *cell gadgets*. Also, each node except the root will be equipped with a *navigation gadget* that signals whether the node is the left or right child of its parent.

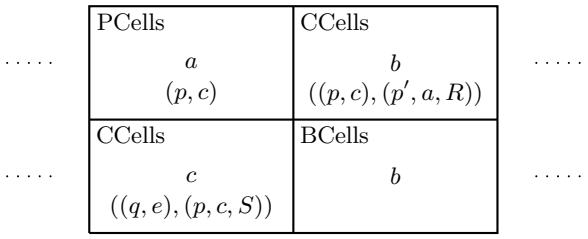
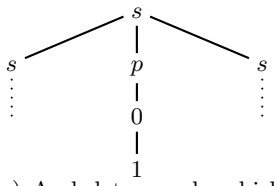
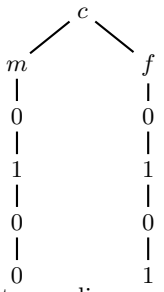


Fig. 3. A representation of a Turing machine transition. The transition used is $t = ((p, c), (p', a, R))$, i.e., the machine is in state p, reads symbol c, writes an a, and moves to the right. The configuration cell encoding the head originally was (the upper left cell) "remembers" the previous state and tape symbol, so that the horizontal constraints can verify that the transition t was actually allowed from the previous configuration.



(a) A skeleton node which is the left child of its parent, with its two skeleton node children and its navigation gadget.



(b) A cell gadget encoding configuration cell 2 in a system with 4 possible configuration cells, where configuration cells number 2 and 4 are not allowed on top of configuration cell 2.

Fig. 4. Gadgets for the CQ validity proof.

More formally, an *encoded configuration tree* is obtained as follows. We start with a full binary tree of height n. The root gets label r and the other nodes label s. The s-labeled nodes are called *skeleton nodes*. A *navigation gadget* is a path (i.e., unary tree) consisting of three nodes. Its root has label p, and the two other nodes 0 and 1, respectively. In an *i-gadget*, for $i \in \{0, 1\}$, the node with label i sits above the one with label $1 - i$ on the path. A skeleton node v which is the left child of its parent is equipped with a 0-gadget (by making v the parent of the p-labeled node in the gadget), while a right child skeleton node gets a 1-gadget; see Figure 4(a).

Each leaf skeleton node (one that has no skeleton node children) is equipped with a *configuration cell gadget*, which is defined as follows. Recall that the configuration cells for M_w are numbered from 1 to k. We describe the gadget for configuration cell i. The root of the gadget has label c (for *cell*) and has two children, labeled m (for *me*) and f (for *forbidden*), respectively. Under the m-labeled node a path of length k is attached. On this path, all nodes have label 0, except the i-th node from the top, which has label 1. Under the f-labeled node, there is also a path of length k. Here, each jth node from the top where $(i, j) \in V(M_w)$ has label 0 while all the positions j

where $(i, j) \notin V(M_w)$ have label 1; see Figure 4(b). This concludes the description of an encoded configuration tree.

The Reduction

The Automaton Definition. The schema is represented by a nondeterministic tree automaton A_{CT}. The automaton should accept a tree t if and only if it satisfies the following properties. For technical reasons, we need t to start at the root with a path of length k to the first CT-labeled node. All nodes on this path have label I and each of them has exactly one child.

1. The subtree rooted at the highest CT-labeled node is a strategy tree. This involves the following steps.
   (a) Each CT-labeled node has exactly one child that is labeled r (i.e., the root of an encoded configuration tree).
   (b) Only configuration cell gadgets that correctly encode configuration cells and vertical constraints of $V(M_w)$ appear.
   (c) Each encoded configuration tree is complete and has the correct height.
   (d) Each skeleton node has a correctly assigned navigation gadget.
2. The CT-labeled nodes on even depth either have zero or two CT-labeled children; the CT-labeled nodes on odd depth either have zero or one CT-labeled child. This reflects the alternating universal and existential moves of Turing Machine M_w.
3. For each CT-labeled node representing a universal configuration, the two child CT-labeled nodes represent two encoded configuration trees with two *different* CCells configuration cells.
4. All horizontal constraints from $H(M_w)$ are satisfied.
5. The highest encoded configuration tree has the start configuration cell $(\text{␣}, ((q_0, \text{␣}), (q_0, \text{␣}, S)))$ as its leftmost configuration cell. Recall that q_0 is M_w's start state, and that M_w's computation starts with an empty tape. This verifies that the computation tree starts with the correct initial configuration of M_w.
6. Each CT-labeled node without CT-labeled node children has a tree attached to it that encodes a final configuration, i.e., its leftmost configuration cell is of the form $(a, ((q_1, b), (q_2, c, M))) \in$ CCells with $q_2 = q_a$. Recall that q_a is the accepting state of M_w and that, upon accepting, M_w moves its tape head entirely to the left. This verifies that each path in the strategy tree leads to an accepting configuration of M_w.

To construct A_{CT}, we construct an automaton for each of the above properties, and use the standard construction for accepting their intersection. It is not hard to see that each property can be checked by a tree automaton whose size is polynomial in the size of the description of M_w — one can essentially hard code each property into an automaton. We briefly describe the automaton A_3 for checking Property 3, as it is technically the most difficult one.

If we think of A_3 as a bottom up automaton, it starts by reading the configuration cell gadgets, and assigns states to their roots; if a gadget represents a configuration cell θ in CCells, A_3 remembers the configuration cell in its state, i.e., it enters a state q_θ. Otherwise, it assigns a neutral state s. When going up to the root of each encoded configuration tree, A_3 simply propagates the state q_θ upwards and checks that the encoded configuration subtree does not contain a second $\theta' \in$ CCells. When A_3 is at the root of an encoded configuration subtree, it propagates q_θ up to the CT-labeled parent.

In the next transition, when going from *two* CT-labeled children to a CT-labeled parent (see also Fig. 1(a)), it tests whether it visited the two CT-labeled children in two *different* states $q_\theta \neq q_{\theta'}$, i.e., whether the attached encoded configuration trees contained different configuration cells $\theta \neq \theta'$ from CCells. Together with the automaton for Property 2 which checks that CT-labeled nodes with one and two CT-labeled node children alternate correctly, this ensures Property 3.

The query. We first define a formula that states that two nodes r_1 and r_2 are roots of two *successive* encoded configuration trees, i.e., encoded configuration trees such that the second encodes the successor configuration of the first.

$$\begin{aligned}\mathrm{Succ}(r_1, r_2) \equiv \exists s_1, s_2 : r(r_1) \wedge r(r_2) \wedge CT(s_1) \wedge CT(s_2) \\ \wedge \mathit{Child}(s_1, r_1) \wedge \mathit{Child}(s_2, r_2) \wedge \mathit{Child}(s_1, s_2)\end{aligned}$$

Next, we define a formula to state that two nodes, x and y, belong to successive encoded configuration trees and are both at level $i > 0$ of their respective encoded configuration tree. Here, $\mathit{Child}^i(x, y)$ abbreviates the formula stating that y can be reached from x by following the Child-axis i times.

$$\Phi_i(x, y) \equiv \exists r_1, r_2 : s(x) \wedge s(y) \wedge \mathrm{Succ}(r_1, r_2) \wedge \mathit{Child}^i(r_1, x) \wedge \mathit{Child}^i(r_2, y)$$

Now we can express that x and y have the property Φ_i and, additionally, that they are either both left children of their parents, or both right children.

$$\begin{aligned}\Psi_i(x, y) \equiv \exists p_x, p_y, t_x, t_y, z : \Phi_i(x, y) \wedge p(p_x) \wedge p(p_y) \wedge 1(t_x) \wedge 1(t_y) \\ \wedge \mathit{Child}(x, p_x) \wedge \mathit{Child}(y, p_y) \wedge \mathit{Child}^+(p_x, t_x) \wedge \mathit{Child}^+(p_y, t_y) \\ \wedge \mathit{Child}^{i+4}(z, t_x) \wedge \mathit{Child}^{i+5}(z, t_y)\end{aligned}$$

With the help of the above predicates, we can now express that two leaf skeleton nodes belong to successive encoded configuration trees and that they correspond to the same position in the configurations. Recall that n is the depth of the encoded configuration trees.

$$\begin{aligned}&\mathrm{SameCell}(s_1, s_2) \equiv \\ &\quad \exists x_1, \ldots, x_{n-1}, y_1, \ldots, y_{n-1} : \bigwedge_{1 \leq i < n-1} (\mathit{Child}(x_i, x_{i+1}) \wedge \mathit{Child}(y_i, y_{i+1})) \\ &\qquad \wedge \mathit{Child}(x_{n-1}, s_1) \wedge \mathit{Child}(y_{n-1}, s_2) \wedge \Psi_n(s_1, s_2) \wedge \bigwedge_{1 \leq i \leq n-1} \Psi_i(x_i, y_i)\end{aligned}$$

Finally, we are ready to define our query Q_{CT} for the Turing Machine M_w, which states that somewhere, a vertical constraint of $V(M_w)$ is violated. Recall that k is the number of configuration cells in CT.

$$\begin{aligned}Q_{CT} \equiv \exists s_1, s_2, t_1, t_2, f_1, m_2, p_1, p_2, z : \mathrm{SameCell}(s_1, s_2) \wedge \mathit{Child}(s_1, t_1) \wedge \mathit{Child}(s_2, t_2) \\ \wedge f(f_1) \wedge m(m_2) \wedge 1(p_1) \wedge 1(p_2) \\ \wedge \mathit{Child}(t_1, f_1) \wedge \mathit{Child}(t_2, m_2) \wedge \mathit{Child}^+(f_1, p_1) \wedge \mathit{Child}^+(m_2, p_2) \\ \wedge \mathit{Child}^{n+k+3}(z, p_1) \wedge \mathit{Child}^{n+k+4}(z, p_2)\end{aligned}$$

For a graphical representation of Q_{CT}, see Figure 5.

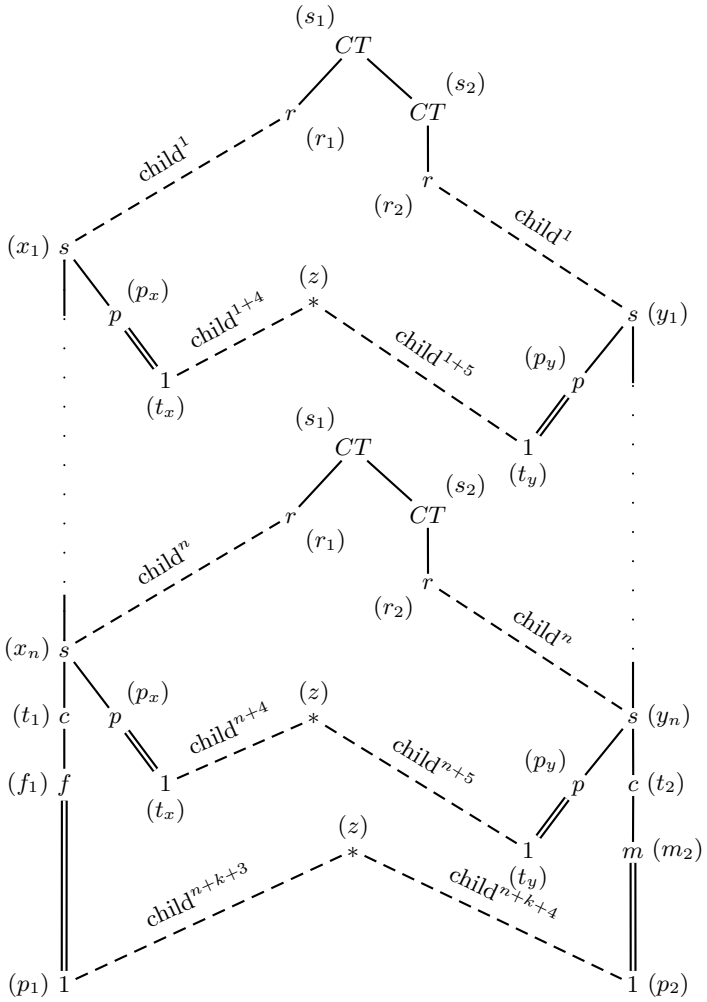


Fig. 5. Graphical representation of Q_{CT} from the proof of Theorem 6. The small labels between braces denote the variable names used in the proof.

C.3 The Lower Bound for DTDs

The main technical observation one has to make is stated in Lemma 18, which was probably first published in [20].

Let A be an NTA. We define the *annotated tree language* of A to be the set of trees in $L(A)$ that are annotated by their accepting runs. More formally, the annotated tree language of A is the set of trees t over $\text{Alpha}(A) \times \text{States}(A)$ where

- $\pi_{\text{Alpha}(A)}(t) \in L(A)$ and
- $\pi_{\text{States}(A)}(t)$ is an accepting run of A on $\pi_{\text{Alpha}(A)}(t)$.

Here, $\pi_{\text{Alpha}(A)}(t)$ denotes the projection of t on $\text{Alph}(A)$, that is, $\pi_{\text{Alpha}(A)}(t)$ is obtained from t by relabeling each label (a, q) to a. (Similarly, $\pi_{\text{States}(A)}(t)$ relabels each (a, q) to q.)

Lemma 18 ([20]). *Given an NTA A, there exists a DTD D_A recognizing the annotated tree language of A. Moreover, D_A can be constructed in quadratic time.*

Theorem 19. *Validity of CQ(Child,Child$^+$) w.r.t. a DTD is 2EXPTIME-complete.*

Proof (Sketch). We describe the changes that have to be made to the proof of Theorem 6. Let $D_{A_{CT}}$ be the DTD accepting the annotated tree language of A_{CT}. Hence, $D_{A_{CT}}$ defines trees over $\text{Alpha}(A_{CT}) \times \text{States}(A_{CT})$. For our reduction, we cannot simply use the set of annotated trees of A_{CT}, as this would require disjunction over alphabet symbols in the definition of the conjunctive query Q_{CT}. Hence, the schema DTD D_{CT} is obtained from $D_{A_{CT}}$ by replacing each rule of the form $(a, q) \to L_{(a,q)}$ where $a \in \{CT, r, s, p, 1, c, m, f\}$ with $(a, q) \to L_{(a,q)} \cdot a$. Hence, we change $L(D_{CT})$ such that, in each of its trees, each (a, q)-labeled node gets an a-labeled child.

It remains to describe how Q_{CT} changes: here, we simply need to replace each atom of the form $a(x)$ (where $a \in \{CT, r, s, p, 1, t, m, f\}$) with $\mathit{Child}(x, y) \wedge a(y)$. The rest of the proof carries through. □

D Proofs of Section 4

Proof of Lemma 7: *There is a polynomial p such that if a CQ Q is satisfiable with respect to an NTA A, then there is a tree $t \in L(Q) \cap L(A)$ and a satisfaction θ of Q on t such that for any variables $x, y \in \mathit{Var}(Q)$, the length of the path from $\theta(x)$ to $\theta(y)$ is at most $p(|A|, |Q|)$.*

Proof. Let t be a tree such that $t \models Q$ and $t \in L(A)$, let θ be a satisfaction of Q on t, let $T = \{\theta(x) \mid x \in \text{Var}(Q)\}$, and let r be an accepting run of A on t. Furthermore, let S be the set of nodes that are least common ancestors of some subset of T of size at least 2. Suppose that there are two nodes $u, v \in T \cup S$ such that u is an ancestor of v, there are no nodes in $T \cup S$ on the path ρ from u to v, and the length of ρ is more than $|\text{States}(A)| \cdot |\Sigma| + 1$. Then there are two distinct nodes $w \neq u$ and $w' \neq v$ on ρ such that w is an ancestor of w', $r(w) = r(w')$, and $\text{lab}^t(w) = \text{lab}^t(w')$. Let t' be the tree obtained from t by replacing the subtree rooted at w with the one rooted at w'. Clearly, r restricted to t' is still an accepting run of A, and θ, restricted to t' is still a satisfaction of Q. This process can be repeated until no nodes $u, v \in T \cup S$ can be found that satisfy the above condition. When this is achieved, the distance, for any $x, y \in \text{Var}(Q)$, between $\theta(x)$ and $\theta(y)$, is at most $1 + |\text{Var}(P)| \cdot (|\Sigma| \cdot |\text{States}(A)| + 1)$. □

Proof of Theorem 8. *Satisfiability of CQs with respect to an NTA is in NP.*

Proof. We can assume w.l.o.g. that A is *reduced*, i.e., each state of A can be used in an accepting run.[7] We know from Lemma 7 that if a query Q is satisfiable with respect to an NTA A, then there is a tree $t \in L(A)$ and a satisfaction θ of Q on t such that for any $x, y \in \mathrm{Var}(Q)$, the distance between $\theta(x)$ and $\theta(y)$ is small (polynomial). This means that if Q is satisfiable with respect to A, then the NP algorithm can guess a polynomial size connected subset t' of nodes of t and a valuation θ of Q on t'. The algorithm also guesses what states an accepting run r of A on t would assign to the nodes in t'. It then verifies that θ is a satisfaction of Q (in polynomial time), and that t' can be extended to a tree in $L(A)$ such that the states assigned to nodes are consistent with the transitions of A. The last check is done as follows. For each node v of t' with label a and it's assigned state q, let $v_1, \ldots, v_n$ be the children of v in t', with labels $a_1, \ldots, a_n$ and assigned states $q_1, \ldots, q_n$, respectively. As A is reduced we only need to test whether there exist transition rules $(q_i, a_i) \to L_i$ in A for each $1 \leq i \leq n$, and that there exist $z_0, \ldots, z_n \in \mathrm{States}(A)^*$ such that there is a transition rule $(q, a) \to L$ in A with $z_0 q_1 z_1 \cdots z_{n-1} q_n z_n \in L$. This last test can be performed in polynomial time by a sequence of n reachability tests on the automaton representing L. □

D.1 Complexity Lower Bounds

For some axes, the result is already known:[8]

Theorem 20 (Wood [22]). *Let A be any element of $\{$Child, Child$^+$, Child$^*\}$. Then Satisfiability for XPath expressions using only axis A w.r.t. a DTD is NP-hard.*

The proof relies on the following Lemma:

Lemma 21 (Wood [22]). *The following problem is NP-hard. Given a (deterministic) regular expression R over alphabet Σ, does $L(R)$ contain a string that contains each Σ-symbol?*

Proof. We reduce from Vertex Cover. Recall that for Vertex Cover we are given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, and ask whether there is a subset $V' \subseteq V$ such that $|V'| \leq k$ and, for each edge $(u, v) \in E$, at least on of u and v belongs to V'. Let $G = (V, E)$ and k be an arbitrary instance of Vertex Cover. We will construct a (deterministic) regular expression R over a finite alphabet Σ such that there is a string in $L(R)$ containing each Σ-symbol if and only if G has a vertex cover of size k or less.

Let $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. For each $1 \leq i \leq n$, let $E_i = \{e_{i,1}, \ldots, e_{i,m_i}\}$, where m_i is the degree of v_i (i.e., the number of edges incident to v_i. The alphabet Σ is given by $E \uplus \{\#\}$, where each $e_i \in E$, $1 \leq i \leq m$, is viewed as a distinct symbol. Let S be the regular expression

$$e_{1,1} \cdots e_{1,m_i} + \cdots + e_{n,1} \cdots e_{n,m_n}.$$

[7] Transforming an NTA to a reduced NTA is in polynomial time.

[8] To the best of our knowledge, the full is unpublished. For the convenience of the referees, we provide Wood's proof, which he kindly provided in personal communication.

Then $R = (S\#)^{k-1}S$, that is, k concatenated occurrences of expression S, separated by #-symbols.

Let $V' = \{v_{j_1}, \ldots, v_{j_k}\}$ be a vertex cover of size k. We find a string $w \in L(R)$ by selecting the j_ith alternation of S for $1 \leq i \leq k$. So w is

$$e_{j_1,1} \cdots e_{j_1,m_{j_1}} \# \cdots \# e_{j_k,1} \cdots e_{j_k,m_{j_k}}.$$

Since V' is a vertex cover for G, w must include every edge in E and hence every symbol in Σ.

Let $w \in L(R)$ be a string which includes every symbol in Σ. String w must be of the form $w = w_1 \# w_2 \# \cdots \# w_k$, where each w_i, $1 \leq i \leq k$, is one of the n strings in $L(S)$. Assume that w_i corresponds to the j_ith alternation in S, that is, $w_i = e_{j_i,1} \cdots e_{j_i,m_{j_i}}$, $1 \leq i \leq k$. Recall that the j_ith alternation in S corresponds to the set of edges adjacent to vertex v_{j_i}. Let $V' = \{v_{j_1}, \ldots, v_{j_k}\}$. Since w contains all symbols in Σ, and the symbols in $\Sigma - \{\#\}$ correspond to edges in E, V' must be a vertex cover for G, and $|V'| \leq k$.

It may be that the regular expressions in this construction are not deterministic. However, the only situation in which the regular expression cannot be rearranged so that it is deterministic is when there are two nodes which are adjacent to each other and nothing else. In this case, the problem is trivial if there are no other nodes. If there are other nodes, then the graph is disconnected in which case the problem can be decomposed. If we assume that the graph is connected then the regular expression can always be made deterministic. □

Given Lemma 21, the rest is easy:

Proof ((of Theorem 20)). By reduction from the problem in Lemma 21. Given a regular expression R over Σ, the DTD accepts all trees of depth 2 in which the children of the root form a string in $L(R)$, and the XPath expression tests whether each Σ-symbol occurs below the root. □

Proof of Theorem 9. *Let Axis be an any element of {Child, Child$^+$, Child*, NextSibling, NextSibling$^+$, NextSibling*, Following}. Then Satisfiability of CQ(Axis) w.r.t. a DTD is NP-hard.*

Proof. Three cases are immediate from the stronger result in Theorem 20. We provide a proof for every Axis in {*NextSibling*, *NextSibling*$^+$, *NextSibling**, *Following*}. For *NextSibling*, we reduce from Shortest Common Superstring, and for all other axes, we reduce from Shortest Common Supersequence. Thereto, let S and k be an input of Shortest Common Superstring (resp., Shortest Common Supersequence). We first provide the proofs for Axis in {*NextSibling*, *NextSibling*$^+$, *Following*}, and then explain how these can be adapted for *NextSibling**. The DTD d for the former three cases has only one rule, namely

$$r \to (a_1 + \cdots + a_n)^k,$$

where $\Sigma = \{a_1, \ldots, a_n\}$. That is, the DTD defines trees of depth 2, in which the root has precisely k children. Let $S = \{b_1^1 \cdots b_1^{n_1}, \ldots, b_m^1 \cdots b_m^{n_m}\}$. Then the query Q is defined as shown in Figure 6. Here, each arrow denotes Axis. It is easy to see that Q is satisfiable w.r.t. d if and only if Shortest Common Superstring (resp., Shortest Common Supersequence) has a solution for S and k if Axis is *NextSibling* (resp., Axis is *NextSibling*$^+$ or *Following*).

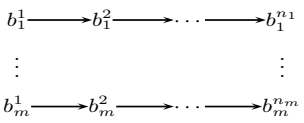


Fig. 6. Gadget for the proof of Theorem 9.

If Axis is *NextSibling**, we adjust the DTD to

$$r \to ((a_1 + \cdots + a_n)\#)^k,$$

where # does not appear in any string in S. The query Q is adapted so that, likewise, between every pair of Σ-symbols, the symbol # must occur. □

E Proofs of Section 5

We first recall some elementary definitions about data words and data trees [3].

Definition 22. – A *data word* $w = w_1 \cdots w_n$ is a finite sequence over $\Sigma \times \Delta$, i.e., each w_i is of the form (a_i, d_i) with $a_i \in \Sigma$ and $d_i \in \Delta$. We denote a_i by $\mathrm{lab}(w_i)$.
– A *data tree* t over Σ has a set of nodes, where every node v has a label $\mathrm{lab}(v) \in \Sigma$ and a data value in Δ.

For a data word $w = w_1 \cdots w_n$ we refer to $\{1, \ldots, n\}$ as the *positions* of w. Here, each position i corresponds to the pair w_i consisting of a label and a data value.

The proofs of the upper bounds make use of the following transformations between data trees over alphabet Σ and (non-data) trees over the alphabet $\Sigma_n = \Sigma \times \{d_1, \ldots, d_n, *\}$, for $n \in \mathbb{N}$.

The set of (non-data) trees over Σ_n is denoted $T(\Sigma_n)$. We first define functions $f_{\not\sim}$ and $f_\sim$, mapping trees from $T(\Sigma_n)$ to data trees. Given a tree $t \in T(\Sigma_n)$, function $f_{\not\sim}$ relabels t and adds data values so as to ensure the following conditions.

1. If $\mathrm{lab}^t(v) = (a, d_i)$, for $a \in \Sigma$ and $i \in \{1, \ldots, n\}$, then node v of $f_{\not\sim}(t)$ has label a and data value d_i.
2. If $\mathrm{lab}^t(v) = (a, *)$, for $a \in \Sigma$, then node v of $f_{\not\sim}(t)$ has label a and a *unique* data value, i.e., a data value that does not appear anywhere else in $f_{\not\sim}(t)$.

Function $f_\sim$, is defined similarly, but all nodes with label $(a, *)$ in t get the same data value d_{n+1}.

Lemma 23. *Given a query Q in $CQ(\sim, \not\sim)$, one can construct NTAs $A^{\sim}_Q$ and $A^{\not\sim}_Q$ in exponential time such that for each Σ_n-tree t it holds*

(a) $t \in L(A^{\sim}_Q)$ *iff* $f_\sim(t) \in L(Q)$, *and*
(b) $t \in L(A^{\not\sim}_Q)$ *iff* $f_{\not\sim}(t) \in L(Q)$.

Proof. The proof is an extension of the proof of Lemma 4, and we use the notation and definitions from that proof. We prove (23 (b)). The proof for (23 (a)) is very similar.

Given $Q \in CQ(\sim, \not\sim)$, we construct an NTA A over Σ_n such that $t \in L(A)$ iff $f_{\not\sim}(t) \in L(Q)$. A state of A has the form (X_a, X_h, X_d, F), where X_a, X_h, X_d are as in the proof of Lemma 4, and $F : \mathrm{Var}(Q) \to \{d_1, \ldots, d_n, *\}$ is a function. Formally, $(X_a, X_h, X_d, F) \in \mathrm{States}(A)$ if (X_a, X_h, X_d) fulfill the conditions (S1)–(S4) in the proof of Lemma 4 and additionally

1. if $x \sim y$ is an atom of Q, then $F(x) = F(y)$ and, if $F(x) = *$, then either both x and y belong to X_h, or none of them do, and
2. if $x \not\sim y$ is an atom of Q, then $F(x) \neq F(y)$ or $F(x) = F(y) = *$ and not both of x and y belong to X_h.

A state (X_a, X_h, X_d, F) is *accepting* if (X_a, X_h, X_d) satisfy conditions (F1)–(F2) in the proof of Lemma 4.

Rules(A)**:** contains all rules of the form

$$\rho = \big((X_a, X_h, X_d, F), (a, \lambda)\big) \to L, \qquad (\dagger)$$

where

(R'1) for each $x \in X_h$, $F(x) = \lambda$ and Q does not contain an atom of the form $b(x)$ with $b \neq a$;
(R'2) for each $(X_a^1, X_h^1, X_d^1, F^1) \cdots (X_a^m, X_h^m, X_d^m, F^m) \in L$, we have $F^1 = \cdots = F^m = F$; and
(R'3) (R2) from the proof of Lemma 4 is satisfied.

For the same reasons as in the proof of Lemma 4, automaton A can be computed in exponential time. The function F increases the state space with a factor of at most $|\mathrm{Var}(q)|^n$.

We show that $L(A) = \{t \in T(\Sigma_n) \mid f_{\not\sim}(t) \models Q\}$. For the inclusion from left to right, take $t \in L(A)$. Consider the valuation θ of Q on t induced by a run of A on t by setting $\theta(x) = v$ if the run assigned state (X_a, X_h, X_d, F) to v, with $x \in X_h$. The proof of Lemma 4 immediately implies that this valuation is a satisfaction of Q' on $f_{\not\sim}(t)$, where Q' is obtained from Q by removing the $\sim$-atoms. We claim that θ is also a satisfaction of Q on $f_{\not\sim}(t)$. Indeed, if $x \sim y$ is a literal of Q, then, for every state used in the run of A on t, $F(x) = F(y) = \lambda$. If $\lambda = d_i$, for some $i \in \{1, \ldots, n\}$ then $\theta(x)$ and $\theta(y)$ in t both have d_i as the second component of their label. Thus $\theta(x)$ and $\theta(y)$ in $f_{\not\sim}(t)$ have the same data value d_i. If $\lambda = *$, then $\theta(x) = \theta(y)$ and we also have that $\theta(x)$ and $\theta(y)$ in $f_{\not\sim}(t)$ have the same data value. Analogously for literals of the form $x \not\sim y$. Hence, $f_{\not\sim}(t) \models Q$.

For the inclusion from right to left, take a $t \in T(\Sigma_n)$ such that $f_{\not\sim}(t) \models Q$. Let θ be a satisfaction of Q on $f_{\not\sim}(t)$. For each $x \in \mathrm{Var}(Q)$, let $H(x)$ be the second component of the label of $\theta(x)$ in t. If $x \sim y$ is a literal of Q, then $H(x) = H(y)$ since $\theta(x)$ and $\theta(y)$ have the same data value in $f_{\not\sim}(t)$. If $H(x) = H(y) = *$ then we must have $\theta(x) = \theta(y)$, since there is no other node in $f_{\not\sim}(t)$ that has the same data value as $\theta(x)$. Analogously for literals $x \not\sim y$.

This means that there is an accepting run r of A on t, such that if $r(v) = (X_a, X_h, X_d, F)$, then $F = H$ and, for each $x \in \mathrm{Var}(Q)$, $x \in X_h$ if and only if $\theta(x) = v$. Keeping in mind that the definition of H ensures that the necessary states are available, the proof of Lemma 4 guarantees the existence of such a run. □

We next show that if $Q \in \mathrm{CQ}(\sim) \cup \mathrm{CQ}(\not\sim)$ the containment test only needs to consider very particular trees.

Let P be a query with variables from $\{x_1, \ldots, x_n\}$ and let t_d be a data tree matching P with satisfaction θ. Then we write $g_n(t_d, \theta)$ for the Σ_n-tree resulting from t_d as follows.

– If $\mathrm{lab}^{t_d}(v) = a$ and $v = \theta(x_i)$, for some i, then v gets label (a, d_j) where j is minimal with $\theta(x_j) \sim v$.

– Otherwise, v gets label $(a, *)$ if $\text{lab}^{t_d}(v) = a$.

Lemma 24. *Let $P, Q \in CQ(\sim, \not\sim)$ and t_d be a data tree such that $t_d \models P$ with satisfaction θ but $t_d \not\models Q$. Then the following hold.*

(a) $f_\sim(g_n(t_d, \theta)) \models P$ and $f_{\not\sim}(g_n(t_d, \theta)) \models P$.
(b) If $Q \in CQ(\sim)$ then $f_{\not\sim}(g_n(t_d, \theta)) \not\models Q$.
(c) If $Q \in CQ(\not\sim)$ then $f_\sim(g_n(t_d, \theta)) \not\models Q$.

Proof. Let P, Q, t_d, θ as stated. It is straightforward that θ is a match of P for both $f_\sim(g_n(t_d, \theta))$ and $f_{\not\sim}(g_n(t_d, \theta))$.

Let us assume $Q \in \text{CQ}(\sim)$. We show (b) by proving that $f_{\not\sim}(g_n(t_d, \theta)) \models Q$ would imply $t_d \models Q$, a contradiction. But this easily follows from the fact that t_d and $f_{\not\sim}(g_n(t_d, \theta))$ only differ on their data values and $\sim$ on $f_{\not\sim}(g_n(t_d, \theta))$ is actually a refinement of $\sim$ on t_d. The proof of (c) similarly uses the fact that $\sim$ on t_d is a refinement of $\sim$ on $f_\sim(g_n(t_d, \theta))$. □

Proof of Theorem 12. *Each of $QC(\sim, \not\sim \mid \sim)$, $QC(\sim, \not\sim \mid \not\sim)$, $QC(\sim \mid \sim, \not\sim)$,w.r.t. an NTA is 2EXPTIME-complete.*

Proof. Hardness is immediate from Theorem 6.

For the upper bound on $\text{QC}(\sim, \not\sim \mid \sim)$ we observe that from Lemma 24 (a) and (b) it follows that if t_d is a counterexample to the containment of P in Q w.r.t. an NTA A then $f_{\not\sim}(g_n(t_d, \theta))$ is a counterexample as well. The upper bound then easily follows by combining $A_P^{\not\sim}$ and $A_Q^{\not\sim}$ as defined in Lemma 23 with the NTA which accepts a Σ_n-tree t iff its Σ-projection is accepted by A.

The upper bound on $\text{QC}(\sim, \not\sim \mid \not\sim)$ follows similarly from Lemma 24 (a) and (c).

For the upper bound on $\text{QC}(\sim \mid \sim, \not\sim)$ it suffices to observe, that if Q uses $\not\sim$ but P does not, then $P \subseteq Q$ holds iff $L(A) = \emptyset$ because in this case trees in which all data values are the same never match Q. □

Lemma 25. *Given $P, Q_1, \ldots, Q_k \in CQ(\sim, \not\sim)$, and an NTA A, queries $P', Q' \in CQ(\sim, \not\sim)$ and an NTA A' can be computed in polynomial time such that $L(P) \cap L(A) \subseteq L(Q_1) \cup \cdots \cup L(Q_k)$ iff $L(P') \cap L(A') \subseteq L(Q')$.*

Proof. This proof is an adaptation of a proof from [16]. Given CQs $Q_1, \ldots, Q_k$ and automaton A, we construct CQs P', Q', and NTA A' such that $P' \subseteq Q'$ w.r.t. A' if and only if $P \subseteq Q_1 \vee \cdots \vee Q_k$ w.r.t. A.

Figure 7 describes a number of query gadgets that we will need in the reduction. The double lines have an extended meaning here; e.g., the double line from T' to Q_1 means that we have a $Child^+(x, y)$ from the variable x carrying T' to *every* variable in the copy of Q_1. The arrows between the T'-labeled nodes in G_Q indicate *NextSibling* predicates. The gadget G_{Q_i} is parameterized by i. Figure 8 shows how copies of the gadgets are put together to form queries P' and Q'. Each copy of a gadget is unique, i.e., for each new copy, the variables are renamed. The automaton A' checks the following properties.

1. There are exactly $2k - 1$ nodes with label S and $2k - 1$ nodes with label T.
2. There are exactly $k \cdot 2(k - 1) + 1$ nodes with label T'.
3. The root has label R and has exactly one child. This child has label S.
4. Each S-labeled node, except one, has one S-labeled child.

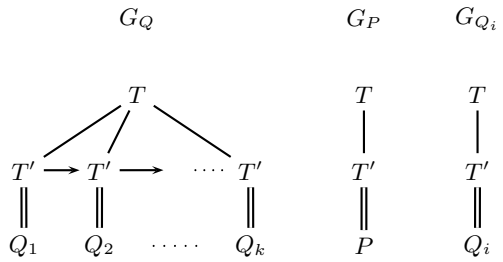


Fig. 7. Gadgets used in the proof of Lemma 25.

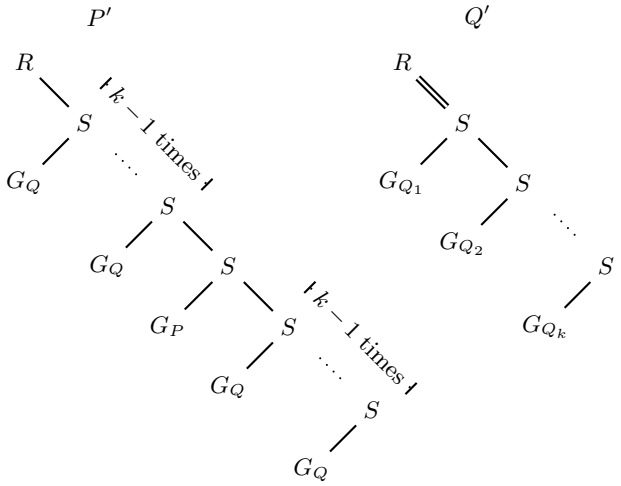


Fig. 8. Queries P' and Q' from the proof of Lemma 25.

5. Each S-labeled node has exactly one child labeled T.
6. Each T-labeled node has exactly k children, each labeled T', except the T-labeled node that is child of the kth S-labeled node, counted from the root. This node has exactly one child, labeled T'. We call this the *distinguished* T-labeled node.
7. Each T'-labeled node has exactly one child.
8. The tree rooted at the grandchild of the distinguished T-labeled node is accepted by A.

Assume that $P \subseteq Q_1 \vee \cdots \vee Q_k$ w.r.t. A. Consider a tree $t \in L(P') \cap L(A')$. Let $s_1, \ldots, s_k, \ldots, s_{2k-1}$ be the S-labeled nodes of t, ordered by increasing distance from the root. For $j \in \{1, \ldots, 2k-1\}$, let t_j be the *tree* rooted in the T-labeled child of s_j. For each $j \in \{1, \ldots, 2k-1\} - \{k\}$, we note that since G_Q matches in the subtree rooted at the T-labeled child of s_j, so does G_{Q_i}, for every $i \in \{1, \ldots, k\}$.

Query G_P must match in t_k. Since t_k only has one T-labeled node, any such matching must assign the topmost variable of G_P to the root of t_k. This means that P must match in the tree t'_k, rooted in the sole grandchild of the root of t_k. Since A must accept t'_k, we conclude that $Q_1 \vee \cdots \vee Q_k$ matches in t'_k, i.e., there is an $i \in \{1, \ldots, k\}$ such that Q_i matches in t'_k. This, in turn, means that G_{Q_i} matches in t_k.

We can now construct a matching of Q in t. The gadgets $G_{Q_1}, \ldots, G_{Q_{i-1}}$ match in $t_{k-i+1}, \ldots, t_{k-1}$, respectively, G_{Q_i} matches in t_k, and $G_{Q_{i+1}}, \ldots, G_{Q_k}$ match in $t_{k+1}, \ldots t_{2k-i}$, respectively.

Assume, on the other hand, that $P \not\subseteq Q_1 \vee \cdots \vee Q_k$ w.r.t. A. Let p be a tree in $(L(P) \cap L(A)) - L(Q_1 \vee \cdots \vee Q_k)$. Let t be a tree in $L(P') \cap L(A')$, and define t_k and t'_k as above. Replace t'_k by p in t. The resulting tree t_p still belongs to $L(P') \cap L(A')$, since p is accepted by A and P matches in p. But since no Q_i, for $i \in \{1, \ldots, k\}$ matches in p, there is no matching of Q' in t_p. Thus $P' \not\subseteq Q'$ w.r.t. A'. □

Proof of Theorem 13. *Validity of a disjunction of CQs with data values w.r.t. an NTA is undecidable.*

Proof. Our proof, inspired by a proof from [17], is by reduction from *Post's Correspondence Problem* (PCP), which is known to be undecidable [18]. An *instance* of PCP over alphabet Σ is a sequence $(w_1, u_1), \ldots, (w_n, u_n)$ of pairs, where $w_i, u_i \in \Sigma^+$, for $i \in \{1, \ldots, n\}$. An instance has a *solution* if there exists an $m \in \mathbb{N}$ and $i_1, \ldots, i_m \in \{1, \ldots, n\}$ such that $w_{i_1} \ldots w_{i_m} = u_{i_1} \ldots u_{i_m}$.

Given an instance $R = (w_1, u_1), \ldots, (w_n, u_n)$ of PCP, we will construct a query Q that is a disjunction of CQs, and an NTA A such that Q is *valid* with respect to A if and only if R has *no* solution.

First, we define the labels to be used by the query. These are the following disjoint sets:

- the root label r and the separator label $\#$;
- the set $I = \{I_1, \ldots, I_n\}$ of index labels; and
- the set Σ.

Let $\Gamma = \{r, \#\} \uplus I \uplus \Sigma$.

The automaton A only accepts unary trees, such that the labels of the tree, read from root to leaf, form a word in the language of the regular expression

$$r \cdot \big((I_1 \cdot w_1) + \cdots + (I_n \cdot w_n)\big)^+ \cdot \# \cdot \big((I_1 \cdot u_1) + \cdots + (I_n \cdot u_n)\big)^+ \cdot \#.$$

Thus, all data trees accepted by A can actually be seen as *data strings*, i.e., strings where each position carries a label and a data value. In order to simplify the terminology in the rest of the proof, we will therefore use standard terminology for strings to reason about these unary trees. The queries we construct will be stated as tree queries, but can be read as queries over strings by interpreting *Child* as the next position predicate and Child^+, Child^* as the transitive and transitive and reflexive closure of *Child*, respectively.

If R has a solution, then there is a string $rw\#u\#$ that is accepted by A such that $w_{|\Sigma} = u_{|\Sigma}$ and $w_{|I} = u_{|I}$. Here, for data string $w = w_1 \cdots w_m$ and $X \subseteq \Gamma$, $w_{|X}$ denotes the Γ-string obtained from $\mathrm{lab}(w_1) \cdots \mathrm{lab}(w_m)$[9] by deleting all positions in $\Gamma - X$. Hence, only the labels in X remain.

The intuition behind our proof is as follows. We encode possible solutions to the PCP by data strings $rw\#u\#$ such that

(ENC1): no data value appears more than twice,
(ENC2): the length of $w_{|I}$ matches the length of $u_{|I}$ and the length of $w_{|\Sigma}$ matches the length of $u_{|\Sigma}$,
(ENC3): two positions with label in I have the same data value iff they correspond to the same position in $w_{|I}$ and $u_{|I}$,
(ENC4): two positions with label in Σ have the same data value iff they correspond to the same position in $w_{|\Sigma}$ and $u_{|\Sigma}$, and
(ENC5): the two occurrences of # have the same data value.

If a data string satisfies the above requirements, we say that it is a *good* encoding. Now, we construct our query Q such that it matches every string accepted by A that is either *(i)* a bad encoding or *(ii)* a good encoding, but does not encode a solution to the PCP problem (i.e., either $w_{|\Sigma} \neq u_{|\Sigma}$ or $w_{|I} \neq u_{|I}$). Hence, if Q is valid w.r.t. A, then no good encoding presents a solution to the PCP problem.

We are now ready to start defining the sub-queries of query Q. (ENC1) and (ENC5) can be easily enforced by

$$\phi_1 \equiv \exists x, y, z\ \mathit{Child}^+(x, y) \wedge \mathit{Child}^+(y, z) \wedge x \sim y \wedge y \sim z$$
$$\phi_2 \equiv \exists x, y\ \#(x) \wedge \#(y) \wedge \mathit{Child}^+(x, y) \wedge x \not\sim y$$

For convenience, we define the binary help-query $\theta(x, y)$ that says[10] that x is in the first half of the string and y in the second.

$$\theta(x, y) \equiv \exists x', x'', y'\ :\ \#(x') \wedge \#(x'') \wedge \#(y') \\ \wedge \mathit{Child}^+(x, x') \wedge \mathit{Child}^+(x', x'') \wedge \mathit{Child}^+(y', y)$$

Our next query, ϕ_3, is satisfied by a string accepted by A iff the second position of the string and the first position after the first appearance of # have different data values. Notice that these positions necessarily have labels from I.

$$\phi_3 \ \equiv\ \exists x_1, x_2, y_1, y_2\ :\ r(x_1) \wedge \#(y_1) \wedge \mathit{Child}(x_1, x_2) \wedge \mathit{Child}(y_1, y_2) \wedge x_2 \not\sim y_2$$

[9] Cfr. Definition 22, p.27.
[10] We are asking readers wondering why we express this property in this awkward fashion for patience until the proof of Theorem 14.

For each pair $(a, b) \in \Gamma$ such that $a \neq b$ we define

$$\phi_{a,b} \equiv \exists x, y \ : \ a(x) \wedge b(y) \wedge x \sim y$$

In order for none of these queries to be satisfied by a string accepted by A, each pair of positions that have the same data value must also have the same label.

For each $i \in \{1, \ldots, n\}$, let $m_i = |w_i|$ and $m'_i = |u_i|$, and define

$$\begin{aligned}\psi_i \equiv \exists x_1, x_2, y_1, y_2 \ : \ & \theta(x_1, y_1) \wedge I_i(x_1) \wedge \mathit{Child}^{m_i+1}(x_1, x_2) \\ & \wedge \mathit{Child}^{m'_i+1}(y_1, y_2) \wedge x_1 \sim y_1 \wedge x_2 \not\sim y_2.\end{aligned}$$

Thus, ψ_i makes sure that if two I-positions have the same data value the same holds for their I-successors.

If A accepts a string $rw\#u\#$, and none of the formulas ϕ_i, $\phi_{a,b}$ or ψ_i is satisfied, then $w_{|I} = u_{|I}$.

It remains to define queries such that if none of them matches a string $rw\#u\#$ accepted by A, then $w_{|\Sigma} = u_{|\Sigma}$. First, we make sure that the first letters in $w_{|\Sigma}$ and $u_{|\Sigma}$ have the same data value:

$$\begin{aligned}\chi_1 \equiv \exists x_1, x_2, x_3, y_1, y_2, y_3 \ : \ & r(x_1) \wedge \#(y_1) \wedge \mathit{Child}(x_1, x_2) \wedge \mathit{Child}(x_2, x_3) \\ & \wedge \mathit{Child}(y_1, y_2) \wedge \mathit{Child}(y_2, y_3) \wedge x_3 \not\sim y_3\end{aligned}$$

Next, we ensure that if positions x_1 and y_1, taken from the first and the second half of the string, respectively, have labels from Σ and the same data value, and their successors also have labels from Σ, then the successors have the same data value. Thus, for each ordered tuple $(a, b, c) \in \Sigma^3$, we define

$$\begin{aligned}\chi^1_{a,b,c} \equiv \exists x_1, x_2, y_1, y_2 \ : \ & \theta(x_1, y_1) \wedge a(x_1) \wedge b(x_2) \wedge c(y_2) \wedge \mathit{Child}(x_1, x_2) \\ & \wedge \mathit{Child}(y_1, y_2) \wedge x_1 \sim y_1 \wedge x_2 \not\sim y_2\end{aligned}$$

We must, however, also allow for the cases that x_1 or y_1, or both of them, are followed by a node with a label from I. Thus, for every (a, b, c) from Σ and every I_i, I_j from I, we get

$$\begin{aligned}\chi^2_{a,b,c,i} \equiv \exists x_1, x_2, x_3, y_1, y_2 \ : \ & \theta(x_1, y_1) \wedge a(x_1) \wedge b(x_3) \wedge c(y_2) \wedge I_i(x_2) \wedge \mathit{Child}(x_1, x_2) \\ & \wedge \mathit{Child}(x_2, x_3) \wedge \mathit{Child}(y_1, y_2) \wedge x_1 \sim y_1 \wedge x_3 \not\sim y_2 \\ \chi^3_{a,b,c,i} \equiv \exists x_1, x_2, y_1, y_2, y_3 \ : \ & \theta(x_1, y_1) \wedge a(x_1) \wedge b(x_2) \wedge c(y_3) \wedge I_i(y_2) \wedge \mathit{Child}(x_1, x_2) \\ & \wedge \mathit{Child}(y_1, y_2) \wedge \mathit{Child}(y_2, y_3) \wedge x_1 \sim y_1 \wedge x_2 \not\sim y_3 \\ \chi^4_{a,b,c,i,j} \equiv \exists x_1, x_2, x_3, y_1, y_2, y_3 \ : \ & \theta(x_1, y_1) \wedge a(x_1) \wedge b(x_3) \wedge c(y_3) \wedge I_i(x_2) \wedge I_j(y_2) \\ & \wedge \mathit{Child}(x_1, x_2) \wedge \mathit{Child}(x_2, x_3) \wedge \mathit{Child}(y_1, y_2) \wedge \mathit{Child}(y_2, y_3) \\ & \wedge x_1 \sim y_1 \wedge x_3 \not\sim y_3.\end{aligned}$$

Our query Q is the disjunction of all the CQs defined above.

We now have to show that $L(A) - L(Q) \neq \emptyset$ if and only if R has a solution.

For the if-direction, let $i_1, \ldots, i_m$ be a solution for R. Let $w = I_{i_1} \cdot w_{i_1} \ldots I_{i_m} \cdot w_{i_m}$ and $u = I_{i_1} \cdot u_{i_1} \ldots I_{i_m} \cdot u_{i_m}$. Let s be a data string with label sequence $rw\#u\#$ such that (ENC1)–(ENC5) hold for s. Clearly, s is accepted by A. We have to show that none of the disjuncts of Q is satisfied by s.

– Queries ϕ_1 and ϕ_2 are not fulfilled by construction.

- Query ϕ_3 is not satisfied by s, since the second position of s and the position after the first # correspond to the same position in $w_{|I} = u_{|I}$ and thus have the same data value.
- None of the queries $\phi_{a,b}$, for $a, b \in \Gamma$, with $a \neq b$ are satisfied, since any two positions in s that have the same data value also have the same label, due to the fact that $w_{|\Sigma} = u_{|\Sigma}$ and $w_{|I} = u_{|I}$.
- None of the queries ψ_i, for $i \in \{1, \ldots, n\}$ are satisfied for the following reason. If position x_1 has label I_i, $\theta(x_1, y_1)$, and $x_1 \sim y_1$, then y_1 also has label I_i, and x_1, y_1 correspond to the same position in $w_{|I} = u_{|I}$. If we go $|w_i| + 1$ steps forward from x_1 and $|u_i| + 1$ steps forward from y_1, we come to a pair x_2, y_2 of positions such that either x_2 and y_2 correspond to the same position in $w_{|I} = u_{|I}$, or they both have label # (they are the first and second #-labeled position, respectively). In either case, x_2 and y_2 have the same data value.
- Query χ_1 is not satisfied because the third position of s and the second position after the first occurrence of # correspond to the same position in $w_{|\Sigma} = u_{|\Sigma}$ and thus have the same data value.
- None of the queries $\chi^1_{a,b,c}$, for $a, b, c \in \Sigma$, are satisfied, for the following reason. If $\theta(x_1, y_1)$, $a(x_1)$, and $x_1 \sim y_1$, then x_1 and y_1 correspond to the same position in $w_{|\Sigma} = u_{|\Sigma}$. Thus $a(y_1)$. If the children x_2 and y_2 of x_1 and y_1 have labels from Σ, then x_2 and y_2 correspond to the same position in $w_{|\Sigma} = u_{|\Sigma}$. Thus they have the same data value.
- None of the queries $\chi^2_{a,b,c,i}$, $\chi^3_{a,b,c,i}$, and $\chi^4_{a,b,c,i,j}$ are satisfied, for very similar reasons.

Thus we can conclude that $s \in L(A) - L(Q)$.

For the only-if-direction, assume that data string s belongs to $L(A) - L(Q)$. We show that s encodes a solution to R. Since s is accepted by A, we know that its label sequence has the form $rw\#u\#$, with $w \in [(I_1 \cdot w_1) + \cdots + (I_n \cdot w_n)]^+$ and $u \in [(I_1 \cdot u_1) + \cdots + (I_n \cdot u_n)]^+$. So, both w and u are non-empty strings. Furthermore, ϕ_1 and ϕ_2 ensure (ENC1) and (ENC5). We need to show that $w_{|I} = u_{|I}$ and $w_{|\Sigma} = u_{|\Sigma}$.

Let $w_{|I} = w^1_{|I} \ldots w^{m_w}_{|I}$ and $u_{|I} = u^1_{|I} \ldots u^{m_u}_{|I}$. Since ϕ_3 is not satisfied by s, we know that the positions corresponding to $w^1_{|I}$ and $u^1_{|I}$ have the same data value. Since none of the formulas $\phi_{a,b}$ is satisfied, this also means that they must have the same label. Thus $w^1_{|I} = u^1_{|I}$. Assume, for $1 \leq k < \min(m_w, m_u)$, that $w^k_{|I} = u^k_{|I}$ and that the two positions of s corresponding to $w^k_{|I}$ and $u^k_{|I}$ have the same data value. Since none of the formulas ψ_i are satisfied, the positions of s corresponding to $w^{k+1}_{|I}$ and $u^{k+1}_{|I}$ also have the same data value. Thus, since none of the formulas $\phi_{a,b}$ is satisfied, $w^{k+1}_{|I} = u^{k+1}_{|I}$. We also have to prove that $m_w = m_u$. Suppose this is not the case. W.l.o.g., we can assume that $m_w < m_u$. We know that $w^{m_w}_{|I} = u^{m_w}_{|I}$. Since none of the formulas ψ_i is satisfied, the node corresponding to the first # in $rw\#u\#$ and the node corresponding to $u^{m_w+1}_{|I}$ in s must have the same data value. Since none of the formulas $\phi_{a,b}$ is satisfied, we must have $u^{m_w+1}_{|I} = \#$, which is a contradiction. Thus we can conclude that $w_{|I} = u_{|I}$. With an analogous induction, using the χ-queries instead of ψ_i, we can show that $w_{|\Sigma} = u_{|\Sigma}$. □

Proof of Theorem 14 *$QC(\not\sim \mid \sim, \not\sim)$ is undecidable.*

Proof. We modify the proof of Theorem 13 and combine it with Lemma 25. In other words, given an instance R of PCP, we construct query Q that is a disjunction of CQs

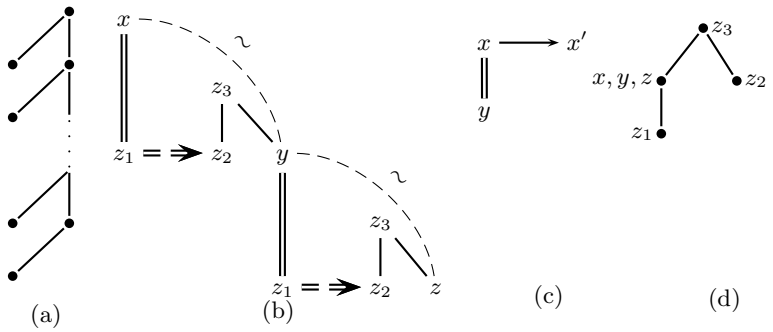


Fig. 9. Trees and queries used in the proof of Theorem 14.

and an automaton A such that Q is valid w.r.t. A iff R has no solution. We can view the constructed problem as a containment problem by setting $P = \text{true}$ and asking whether $L(P) \cap L(A) \subseteq L(Q)$. To remove the disjunction in Q, we use Lemma 25. If we do this straightforwardly, we would obtain CQs P', Q' and an NTA A' such that $L(P') \cap L(A') \subseteq L(Q')$ iff R has no solution. Since the construction of P' in Lemma 25 uses copies of the disjuncts of Q, we would not, however, be sure that P' doesn't use $\not\sim$, even though P clearly doesn't. Luckily, it turns out that a slight modification of the construction in the proof of Theorem 13 allows us to in turn adapt the construction of the proof of Lemma 25 such that $\sim$-atoms in P' are avoided.

In other words, from Q and A in the proof of Theorem 13, we construct queries $P_{\not\sim}$, $Q_{\sim,\not\sim}$, and NTA A' such that Q is valid w.r.t. A iff $L(P_{\not\sim}) \cap L(A') \subseteq L(Q_{\sim,\not\sim})$.

We first describe how we modify the encoding of solution candidates for R from the proof of Theorem 13. We modify the unary trees used in that proof as follows. Each node gets a new extra leftmost child. The new nodes get their label from their parent node. The "old" nodes all get a new "blank" label (see Figure 9(a)). Clearly, the automaton A can be adapted to take care of this new shape of trees.

Now, we change the disjunction Q into a new query Q', which is also a disjunction of conjunctive queries. In each disjunct Q_i of Q we first add a conjunct $\text{blank}(z)$ for each variable z, stopping the original variables from binding to the newly added extra children. Afterwards, each atomic formula $a(x)$, for $a \in \Gamma$ is replaced by $\exists x' \mathit{Child}(x, x') \wedge a(x')$, where x' is a fresh variable. I.e., the new nodes can only be matched with variables in the new sub-formulas for $a(x)$. As a last change, we replace the adapted ϕ_1 in Q', i.e., we replace $\mathit{Child}^+(x, y)$ by

$$\exists z_1, z_2, z_3 \ : \ \mathit{Child}^+(x, z_1) \wedge \mathit{Following}(z_1, z_2) \wedge \mathit{Child}(z_3, z_2) \wedge \mathit{Child}(z_3, y),$$

and $\mathit{Child}^+(y, z)$ by the same kind of gadget. The resulting query is depicted in Figure 9(b) and is called ϕ_1'. On the intended trees resulting from a path by the above encoding (Fig. 9(a)) this does not change the semantics.

The NTA A' and the query $Q_{\sim,\not\sim}$ are obtained from applying the proof of Lemma 25 to the adapted automaton A and to the query Q'.

We still need to define $P_{\sim}$. Thereto, let P' be the query that would be obtained from applying the proof of Lemma 25 to $Q' = Q_1' \vee \cdots \vee Q_m'$ and A. From the construction of Lemma 25, we have that P' contains several occurrences of the gadget $G_{Q'}$, which can contain data equalities. We will change $G_{Q'}$ to $G_{Q'}^{\not\sim} \in \text{CQ}(\not\sim)$ such that, for each

$i \leq m$, $G^{\not\sim}_{Q'} \subseteq Q'_i$. Hence, we replace sub-queries Q'_j in $G_{Q'}$ containing $\sim$-atoms by sub-queries $Q^{\not\sim}_j$ without $\sim$-atoms (but possibly with $\not\sim$-atoms) such that the inclusion property is preserved. The main idea for achieving this is as follows: We make sure that, if $x \sim y$ is an atom of Q'_i, then, for every tree in $t \in L(Q^{\not\sim}_i)$, there is a satisfaction θ of Q'_i on t such that $\theta(x) = \theta(y)$.

For example, if Q'_j is one of the $\phi_{a,b}$-formulas from Theorem 13, then

$$Q^{\not\sim}_j \equiv \exists x, y, y' \ : \ \mathit{Child}(x, y) \wedge \mathit{Child}(x, y') \wedge a(y) \wedge b(y')$$

In any tree satisfying $Q^{\not\sim}_j$, the variables x, y from Q'_j can bind to the same node, while this is impossible in a tree accepted by the updated A.

If Q'_j is some ψ_i we set

$$Q^{\not\sim}_j \ \equiv \ \exists x_1, x_2, y_2 \ : \ \theta(x_1, x_1) \wedge I_i(x_1) \wedge \mathit{Child}^{m_i+1}(x_1, x_2) \wedge \mathit{Child}^{m'_i+1}(x_1, y_2) \wedge x_2 \not\sim y_2.$$

Note that $\theta(x_1, x_1)$ can be satisfied by strings containing the symbol # three times. The χ-formulas can be handled similarly.

Concerning the sub-query ϕ_1 of Q' we have used a little trick, by changing the sub-query in Q' before we defined the corresponding part of $G_{Q'}$. The corresponding sub-query $Q^{\not\sim}_j$ in $G^{\not\sim}_{Q'}$ is defined as $\phi^{\not\sim}_1 \equiv \exists x, y, x' : \mathrm{blank}(x) \wedge \mathit{Child}^+(x, y) \wedge \mathit{NextSibling}(x, x')$ (Fig. 9(c)). Note that if some v matches x in a satisfaction of this formula then it also matches x, y, and z in ϕ'_1 (i.e., Q'_j). Indeed, consider the tree in Figure 9(d), which is a tree satisfying $\phi^{\not\sim}_1$. If we match x of ϕ'_1 onto the left child of the root, we can also match y and z of ϕ'_1 onto the left child of the root. The variable assignments matching ϕ'_1 to the tree are depicted in Figure 9(d). This concludes the definition of $P_{\not\sim}$.

We now have that Q is valid w.r.t. A iff $L(P_{\not\sim}) \cap L(A') \subseteq L(Q_{\sim,\not\sim})$. Validity of unions of CQs with $\sim$ and $\not\sim$ can thus be reduced to $\mathrm{QC}(\not\sim \mid \sim, \not\sim)$ and we can conclude that the latter is undecidable. □

E.1 Containment without Schema Information

The following proofs concern containment without schema information. As, in Section 2.2, we only defined the semantics of Conjunctive Queries w.r.t. trees using a finite labeling alphabet, we need to extend this definition here to trees where the labeling alphabet is not necessarily fixed in advance.

That is, Σ_{inf} is now a fixed but countably infinite set of labels. A Σ_{inf}-tree t is a relational structure, as before, over a finite number of unary labeling relations $a(\cdot)$, binary relations $\mathit{Child}(\cdot, \cdot)$, and $\mathit{NextSibling}(\cdot, \cdot)$. Here, we now have that each $a \in \Sigma_{\mathrm{inf}}$.

Conjunctive queries over Σ_{inf}-trees are defined completely analogously to conjunctive queries over Σ-trees.

For schema-less containment, we need a new Lemma in spirit of Lemma 25.

Lemma 26. *Given CQ($\sim, \not\sim$) over Σ_{inf}-trees $P, Q_1, \ldots, Q_k$, the problem of determining whether $L(P) \subseteq L(Q_1) \cup \cdots \cup L(Q_k)$ w.r.t. A is polynomial time reducible to containment for CQ($\sim, \not\sim$) over Σ_{inf}-trees P' and Q'.*

Proof. The proof is analogous to the proof of Lemma 3 in [16] (and quite similar to the proof of Lemma 25). □

In the next theorem, we consider containment over Σ_{inf}-trees.

Proof of Theorem 15: *Containment for CQ($\sim, \not\sim$) is undecidable, even without a schema.*

Proof. The proof builds on the proof of Theorem 13. Given an instance $(w_1, u_1), \ldots, (w_n, u_n)$ of PCP, we construct queries P and Q, where P is a CQ($\sim, \not\sim$) and Q is a disjunction of CQ($\sim, \not\sim$), such that $P \subseteq Q$ if and only if $(w_1, u_1), \ldots, (w_n, u_n)$ has *no* solution.

We can assume that the strings $w_1, u_1, \ldots, w_n, u_n$ are Σ-strings. As before, we define $\Gamma = \{r, \#\} \uplus I \uplus \Sigma$.

In the absence of an NTA, there are two additional aspects that we have to use the queries to take care of:

(1) The structure of any possible counterexample, i.e., it should be a string matching the regular expression

$$r \cdot [(I_1 \cdot w_1) + \cdots + (I_n \cdot w_n)]^+ \cdot \# \cdot [(I_1 \cdot u_1) + \cdots + (I_n \cdot u_n)]^+ \cdot \#.$$

(2) Since we no longer have a finite predetermined set Σ of possible node labels, we cannot, as in the proof of Theorem 13 ensure that two nodes have the same label by using a disjunction over all pairs of non-equal labels. Thus we have to use data values to *encode* labels.

We first describe how to achieve (1). The query P is simple:

$$\begin{aligned} P \equiv \exists x, x_1, y, y_1, z \ : & \, r(x) \wedge \#(y) \wedge \#(z) \\ & \wedge \mathit{Child}(x, x_1) \wedge \mathit{Child}^+(x_1, y) \wedge \mathit{Child}(y, y_1) \wedge \mathit{Child}^+(y_1, z). \end{aligned}$$

As in the proof of Theorem 13, we use the query Q to

- capture bad encodings of possible solutions for the PCP and
- find mistakes in all good encodings of possible solutions for the PCP.

To make sure that a counter example does not branch, we add the following query as a disjunct to Q.

$$\exists x, y \ : \ \mathit{NextSibling}(x, y)$$

To ensure that only the first position has label r we also add

$$\exists x, y \ : \ \mathit{Child}(x, y) \wedge r(y).$$

For every $i \in \{1, \ldots, n\}$ we must make sure that I_i is followed by w_i (if in the first half of the solution string) followed by I_j (for some j) or $\#$. To this end, we write one query for every possible deviation from this pattern. I.e., for every string s in $\Gamma^{|w_i|} - \{w_i\}$ we write a query that matches the pattern $I_i \cdot s$, and for every $a \in \Sigma$, we write a query that matches the pattern $I_i \cdot w_i \cdot a$.

Next, we describe how to achieve (2). Assume that $i_1, \ldots, i_m$ is a solution to $(w_1, u_1), \ldots, (w_n, u_n)$, and consider the data string

$$S = r I_{i_1} w_{i_1} \ldots I_{i_m} w_{i_m} \# I_{i_1} u_{i_1} \ldots I_{i_m} u_{i_m} \#$$

annotated with data values as in the proof of Theorem 13. Construct a new string, S', that encodes the solution in a slightly more involved way. The first position of S' has label r and the same data value as the first position in S. We describe how we

encode a position $j > 1$ of S in S'. Let $k = \lceil \log_2 |\Gamma| \rceil$. Let $num : \Gamma \to \{1, \ldots, |\Gamma|\}$ be an injective function that assigns a unique number to each symbol in Γ. We will actually use a sequence $j_1, \ldots, j_{k+2}$ of positions in S' to encode j. The first position, j_1, in this sequence has the same data value as the first position of S' (labeled r) and is called a *marker position*, as it marks the start of an encoding of a position j of S. The marker positions of S' will be the only positions that has the same data value as the first position of S'. The second position, j_2, is called a *data position*, and carries the same data value as position j in S. This data value will serve two purposes. First, it will be used just as the data value of position j in S was used in the proof of Theorem 13. Second, it will function as a reference point for positions $j_3, \ldots, j_k$. More precisely, we will interpret the data values of the positions $j_3, \ldots, j_k$ as a binary encoding of a Γ-symbol. That is, if position j_ℓ, for $\ell \in \{3, \ldots, k+2\}$ has the same data value as j_2, we will interpret this as a 1-bit. If the two data values are different, we will interpret this as a 0-bit. Let $a \in \Gamma$ be the label of position j in S. Then the actual data values for $j_3, \ldots, j_{k+2}$ in S' are chosen in such a way that the binary string they encode represents $num(a)$. Since it may be the case that $\lceil \log_2 |\Gamma| \rceil > \log_2 |\Gamma|$ we must make sure that the the bit strings of length k that do not represent a symbol in Γ do not appear in a counter example. This is easily achieved by, for each such bit string, adding a disjunct to Q that matches it.

To ensure that only marker positions have the same data values as the first position in S', we add disjuncts to Q that match if one of the following conditions is not satisfied (i.e., we want to ensure that the following three properties hold).

1. The second position in S' has the same data value as the first position in S'.
2. If x is not the first position of S' but has the same data value as the first position of S', then position $x + k + 2$ has the same data value as x.
3. If x is not the the first position of S' but has the same data value as the first position of S', then, for each $j \in \{1, \ldots, k+1\}$, position $x + j$ has a different data value than x.

It is easy to see that we can encode the negation of each of these properties with a disjunction of conjunctive queries. For instance, for case 3 this would be the disjunction of the formulas

$$\exists x, y, x_j : \text{NotFirst}(x) \wedge \text{First}(y) \wedge x \sim y \wedge \mathit{Child}^j(x, x_j) \wedge x \sim x_j$$

for each $j \in \{1, \ldots, k+1\}$, where $\text{First}(y)$ and $\text{NotFirst}(x)$ are predicates that are satisfied on the first position (resp., not the first position) of S'. Predicate First is a CQ that tests whether y is labeled r, and predicate NotFirst tests whether x has an ancestor.

We can now, for every $a \in \Gamma$, define the predicate $a'(x)$ that will be true of position x if x is the marker position of a sequence that encodes a. This predicate simply checks that x has the same data value as the first position and that the bit-string encoded below x represents $num(a)$. Also, we can define $\sim'$, which is true of two positions x, y if both are marker positions and the data values of the two data positions following them are the same. Predicate $\not\sim'$ is defined symmetrically.

We now simply replace all occurrences of $\sim$ in Q by $\sim'$, $\not\sim$ by $\not\sim'$, $a \in \Gamma$ by a', and *Child* by Child^{k+2}. □

F Proofs of Section 6

Proof of Corollary 16: *In general, there does not exist an exponential-size nondeterministic tree automaton recognizing $\overline{L(Q)}$, where Q is a CQ(Child, Child$^+$).*

Proof. Towards a contradiction, assume that, for every conjunctive query Q, there exists an exponential-size NTA A_Q for $\overline{L(Q)}$. This means that, if there is a counterexample for the containment problem $P \subseteq Q$ w.r.t. NTA A, there always exists a counterexample of exponential depth. However, this would imply, according to the proof of Theorem 6, every EXPSPACE alternating Turing Machine has an accepting computation tree of at most exponential depth, which is a contradiction. □

G Further Remarks

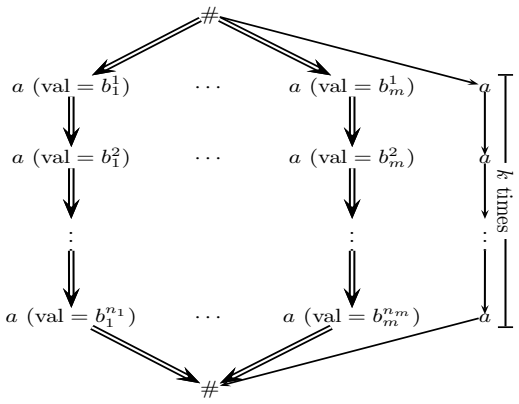


Fig. 10. Query Q for the proof of Lemma 27.

Lemma 27. *Query satisfiability for queries with structural constraints, Value Based Constraints (VBCs) and no wildcards is NP-hard.*

Proof. This follows from an easy adaptation of the proof of Lemma 9. That is, we reduce from Shortest Common Supersequence. Thereto, let S and k be an input of Shortest Common Superstring (resp., Shortest Common Supersequence). Let $S = \{b_1^1 \cdots b_1^{n_1}, \ldots, b_m^1 \cdots b_m^{n_m}\}$. Then the query Q is defined as shown in Figure 10. Here, each single arrow denotes the *Child*-axis, and each double arrow denotes the *Child*$^+$-axis. The confluency in the bottom #-labeled node is obtained via *structural constraints*, which allow to identify nodes (see [14]). All nodes, apart from the two #-labeled nodes bear the alphabet label a. Finally, the val $= x$ equations denote the *value-based constraints* — they say that the value at the current node must be equal to x.

It is easy to see that Q is satisfiable if and only if Shortest Common Supersequence has a solution for S and k. The idea is that the common supersequence for S must be formed in the data values for the a-labeled nodes at the right hand side of Figure 10. □