Noname manuscript No.
(will be inserted by the editor)

Conjunctive Query Containment over Trees using Schema Information

Henrik Björklund · Wim Martens · Thomas Schwentick

Received: date / Accepted: date

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 with respect to a schema, in both cases where the schema is given as a DTD or as a tree automaton. Furthermore, we show that satisfiability for conjunctive queries with respect to 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 with respect to 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 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

A preliminary version of this work was presented at the 33rd International Symposium on Mathematical Foundations of Computer Science.

Henrik Björklund
Umeå University, Department of Computing Science

Wim Martens
Universität Bayreuth, Institut für Informatik

Thomas Schwentick
Technische Universität Dortmund, Department of Computer Science

of Chandra and Merlin [14], 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 [14], and the Constraint Satisfaction Problem (CSP) in Artificial Intelligence [31].

The rise of semi-structured data and XML initiated the investigation of conjunctive queries over trees [28]. 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. [28], 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.

For conjunctive queries over trees, in contrast to the relational setting, 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 [7].

Conjunctive query containment over trees is therefore investigated directly in [7], but was also treated more implicitly in the form of XPath 2.0 static analysis in, e.g., [29,32,44]. We elaborate on the relation with these papers later. In a nutshell, XPath 2.0 puts syntactic constraints on conjunctive queries which sometimes limit them. The results in [7] 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 [7] 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 worst case 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 with respect to a DTD, is already 2EXPTIME-hard for queries using only the *Child* and *Child*$^+$ axes. We even strengthen this result to show that the validity problem of a conjunctive, positive fragment of XPath 2.0 queries with respect to a DTD is 2EXPTIME-hard. 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

[38] to (at most) EXPTIME [40,37,8]. 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.

The picture again changes dramatically when we consider satisfiability instead of containment. Even for the most general conjunctive queries with respect to Relax NG schemas, the satisfiability problem is in NP. Unfortunately it is also NP-hard already 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.

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; see Section 7.

The remainder of the article is structured as follows. After introducing the basic material that will be used throughout the paper (Section 2), we prove that validity and containment of conjunctive queries over trees is 2EXPTIME-complete with respect to schemas in Section 3. We make a brief excursion to satisfiability of conjunctive queries over trees with respect to schema information in Section 4 and conclude the technical part of the paper by a study of containment of queries with data value comparisons in Section 5. In Section 6 we relate our results with XPath with *path intersection.* Section 7 explains how all our results carry over from boolean queries to higher-arity queries. We end by a discussion on related work, a remark on a result of Lakhsmanan (Section 8) and then move to the conclusions (Section 9).

2 Preliminaries

We consider 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 assume some infinite set of labels that contains all labels throughout the paper, but in most scenarios there is some finite alphabet Σ from which the labels of a tree or a query are chosen. We view a tree t as a relational structure with unary labeling relations $a(\cdot)$, 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. We assume that each node carries exactly one label and write $\text{lab}^t(u)$ for the unique label a such that $a(u)$ holds in the tree t. We often omit t from this notation when t is clear from the context.

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 further consider a binary relation *Following*, corresponding to the *Following* axis of XPath, which can be defined given the other relations by the formula

$$\mathit{Following}(z_1, z_2) = \exists x \exists y : \mathit{Child}^*(x, z_1) \wedge \mathit{NextSibling}^+(x, y) \wedge \mathit{Child}^*(y, z_2).$$

We refer to the binary relations above as *axes*. We denote the set of nodes of a tree t by $\text{Nodes}(t)$. By $\text{root}(t)$ we denote the root node of t.

2.1 Conjunctive Queries over Trees

Let $X = \{x, y, z, \ldots\}$ be a set of variables. A *conjunctive query* (CQ) over an alphabet Σ is a positive existential first-order formula without disjunction over a finite set of unary predicates $a(x)$ with $a \in \Sigma$, and the binary predicates *Child*, Child^+, Child^*, *NextSibling*, $\mathit{NextSibling}^+$, $\mathit{NextSibling}^*$, and *Following*.[1] 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 conjunctive queries. 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 $\text{Var}(Q)$. We use $\text{CQ}(R_1, \ldots, R_k)$ or $\text{CQ}(\mathcal{R})$ (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. [28]. That is, let Q be a CQ, and t a tree. A *valuation* of Q on t is a total function $\theta : \text{Var}(Q) \to \text{Nodes}(t)$. 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.[2] We denote the complement of $L(Q)$ by $\overline{L(Q)}$.

As usual, we refer by UCQ to the class of disjunctions (or: unions) of conjunctive queries with the same conventions regarding parameters as for CQ.

[1] We do not require CQs to be in prenex normal form. However, all formulas that we construct in the paper can be put in prenex normal form by simply renaming the variables and moving the quantifiers.

[2] 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, the set $L(Q)$ also consists of trees over alphabet Σ. In the rare cases where we consider trees without schema information, we state this explicitly.

2.2 Schemas

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

Definition 1 *A* Document Type Definition (DTD) *over an alphabet Σ is a triple $D = (\mathit{Alpha}(D), \mathit{Rules}(D), \mathit{start}(D))$ where $\mathit{Alpha}(D) = \Sigma$, $\mathit{start}(D) \in \Sigma$ is the start symbol and $\mathit{Rules}(D)$ 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)* $\text{lab}^t(\text{root}(t)) = \text{start}(D)$ and, *(ii)* for every $u \in \text{Nodes}(t)$ with label a and n children $u_1, \ldots, u_n$ from left to right, there is a rule $a \to R$ in $\text{Rules}(D)$ such that $\text{lab}^t(u_1) \cdots \text{lab}^t(u_n) \in L(R)$. By $L(D)$ we denote the set of trees satisfying D.

We abstract from Relax NG schemas [16] by unranked tree automata, which are formally defined as follows:

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

DTDs are strictly less expressive than Relax NG schemas. For instance, Relax NG schemas are powerful enough to require that two a-labeled nodes have differently labeled children, whereas DTDs cannot express this. For more precise characterizations on the expressiveness of DTDs, Relax NG schemas, and XML Schema (for which the expressiveness lies between DTD and Relax NG) we refer to [36].

For simplicity, we denote the regular languages R in DTD or NTA rules by regular expressions. For our complexity results, it does not matter whether the languages R are represented by regular expressions or nondeterministic finite word automata.

A *run* of A on a tree t is a labeling $r : \text{Nodes}(t) \to \text{States}(A)$ such that, for every $u \in \text{Nodes}(t)$ with label a and children $u_1, \ldots, u_n$ from left to right, there exists a rule $(q, a) \to R$ such that $r(u) = q$ and $r(u_1) \cdots r(u_n) \in L(R)$. Note that when u has no children, the criterion reduces to $\varepsilon \in L(R)$, where ε denotes the empty word. 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.

2.3 Our Problems of Interest

We are primarily concerned with the following three decision problems:

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 the above problems are instances of the containment problem or its complement. More precisely, validity of Q is testing whether $L(\text{true}) \cap L(S) \subseteq L(Q)$ and satisfiability for Q is testing whether $L(Q) \cap L(S) \not\subseteq L(\text{false})$.

We note that, for all these algorithmic problems, the alphabet Σ of possible labels is determined by the schema S.

3 Validity and Containment

In this section we prove that validity of conjunctive queries with respect to schemas is 2EXPTIME-complete. This holds when the schema is given as a non-deterministic tree automaton, as well as a DTD.

3.1 Complexity Upper Bounds

We first settle the upper bound for the containment problem. This is achieved through a standard translation of existential first-order logic into NTAs (see, e.g., [45] and, with a discussion of the upper bound, [24, Theorem 4.2]). To remain self-contained, we describe the construction in detail.

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.*

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_c, X_d), where X_a, X_c, 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_c is the set of variables that A guesses to be placed *on the current node*, and
- X_d is the set of variables that A guesses to be placed on *descendants of the current node.*

Since 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_c, X_d), the pairwise intersections of X_a, X_c, and X_d are empty.

In order to define A formally, we specify States(A), Final(A), and Rules(A).

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

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

Final(A): A state (X_a, X_c, X_d) of A is in Final(A) if and only if

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

Rules(A): contains all rules of the form

$$\rho = \big((X_a, X_c, X_d), a\big) \to R,$$

where

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

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) $L(A) = L(Q)$.

Concerning (1), it is clear that States(A) and Final(A) can be computed in time exponential in $|Q|$. For Rules(A), we prove that we can compute a non-deterministic finite word automaton (NFA) N that accepts, for every $(X_a, X_c, X_d) \in$ States(A) and $a \in$ Alpha(A), the language $L(R)$ in the rule

$$\rho = \big((X_a, X_c, X_d), a\big) \to R.$$

Since the alphabet of N is States(A), we know that every symbol it reads satisfies (S1)–(S4). Furthermore, (R1) does not need to be checked by N, but rather by the algorithm that constructs A, when deciding whether or not to define a transition rule of the form of ρ. 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 word. 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 $(X_c^\cup, X_d^\cup, Y_{ns}, Y_{ns*})$, where the components are defined as follows. Suppose that N has read the prefix

$$(X_a^1, X_c^1, X_d^1) \cdots (X_a^k, X_c^k, X_d^k)$$

of a word $(X_a^1, X_c^1, X_d^1) \cdots (X_a^n, X_c^n, X_d^n)$. Then

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

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

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

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

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

Finally, N accepts if

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

(2) A simple induction on the depth of a tree is sufficient to show that A recognizes the language $L(Q)$ defined by the query. In particular, for every tree t, each satisfaction of Q on t induces an accepting run of the automaton A on t. □

It is now easy to derive the following theorem, showing that containment of conjunctive queries with respect to NTAs is in 2EXPTIME. We note that this upper bound is not really new. It can be obtained by composing the exponential translation of [28] from CQs to Core XPath and the polynomial time translation of [44] 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 [46]. However, we give this self-contained proof here, as the construction of Lemma 4 will be re-used in Lemma 18 later in the paper.

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 (see, e.g., Theorem 19(1) in [34]). Let A be the schema NTA and let P and Q be the queries. According to Lemma 4, we can compute in exponential time two automata A_P and A_Q such that $L(A_P) = L(P)$ and $L(A_Q) = L(Q)$. 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. $\square$

3.2 Complexity Lower Bounds

In this section, we prove the following result.

Theorem 6 *Validity of CQ(Child,Child$^+$) with respect to tree automata is 2EXPTIME-complete.*

Furthermore, 2EXPTIME-hardness holds even if the tree automaton has a constant-size alphabet (we use nine different labels in the proof).

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 [13].

The overall idea of our proof is as follows. Let M be a given alternating Turing machine (ATM) M and w a word of length n. For technical reasons, we first construct, in polynomial time, an ATM M_w which accepts the empty word if and only if M accepts w. From M_w we construct an NTA A_{CT} that checks most important properties of (suitably encoded) computation trees t of M_w, except their consistency with respect to the transition relation of M_w. Furthermore, we construct a Boolean query Q_{CT} that is satisfied by a computation tree t in $L(A_{CT})$ if and only if the transition relation of M_w is *not* respected by t. Altogether, Q_{CT} is valid with respect to A_{CT}, if and only if 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.

3.2.1 Alternating Turing Machines

An *alternating Turing machine (ATM)* [13] is a tuple $M = (Q, \varSigma, \varGamma, \delta, q_0)$ where $Q = Q_\forall \uplus Q_\exists \uplus \{q_a\} \uplus \{q_r\}$ is a finite set of states partitioned into the *universal states* $Q_\forall$, the *existential states* $Q_\exists$, an *accepting state* q_a, and a *rejecting state* q_r. The (finite) input and tape alphabets are $\varSigma$ and $\varGamma$, respectively, with $\varSigma \subseteq \varGamma$. 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 $(Q \times \varGamma) \times (Q \times \varGamma \times \{L, R, S\})$. 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 word $w_1 q w_2$ where $w_1, w_2 \in \varGamma^*$ 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 as for standard Turing Machines. For configurations κ_1, κ_2, and transition τ, we denote by $\kappa_1 \vdash_\tau \kappa_2$ that M can move from κ_1 to κ_2 by performing transition τ. When $q = q_a$ or $q = q_r$, we say that $w_1 q w_2$ is a *halting configuration.* We can assume without loss of generality that no halting configuration has a successor configurations, and that each non-halting configuration has precisely two *different* successor configurations. Furthermore, we can assume without loss of generality that each halting configuration is of the form qw, i.e., M moves its head to the beginning of the tape before halting.

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 exactly one child v, which 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 $a \in \varGamma$, 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 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 [13].

In our reduction, we will work with an ATM *without input.* In order to do this, given an ATM M whose word problem is 2EXPTIME-complete, and

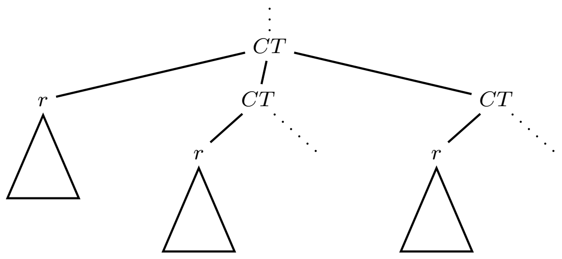


Fig. 1 A part of an encoded configuration tree. The CT-labeled nodes define the structure of the actual configuration tree of M_w, while the subtrees with root label r encode the actual configurations of M_w.

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.

3.2.2 The Encoding

The NTA we construct from M_w will recognize encoded computation trees of M_w. We now descibe how this encoding works, i.e., how we represent computation trees and configurations in the reduction.

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

– the root(t_u) is labeled CT;
– the leftmost child of root(t_u) is labeled r (and is root of the subtree that encodes the actual configuration at u in t); and
– for each child u_i of u, root(t_u) has a subtree enc(t/u_i) where t/u_i denotes the subtree of t rooted at u_i.

Hence, Figure 1 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 encode a configuration of M_w as a sequence of 2^n *configuration cells.* Such a cell contains the content of a tape cell of M_w, plus some additional information. In particular, the configuration cell that encodes the tape cell currently visited by M_w also contains information about the current state. In addition, we need some information that will be used to verify that the transitions of M_w are respected. Before describing the details, we need the following convention for talking about transitions. If $\tau = ((q_1, a), (q_2, b, m))$ is a transition of M_w, then we say that q_1 is the *from-state* of τ, that a is the *read-symbol*, q_2 is the *to-state*, b is the *write-symbol*, and m is the *direction*.

We use three types of configuration cells:

- 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 were not 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. (In the initial configuration, we can use an arbitrary element from δ.)
- The set PCells of *previous tape-head cells* is equal to $\Gamma \times (Q \times \Gamma \times \{L, R\})$. These cells represent tape cells that were visited by the tape head in the previous configuration, and not in the current one. The letter from Γ represents the current cell content, while the triple from $Q \times \Gamma \times \{L, R\}$ represents the machine state in the previous configuration, the cell content in the previous configuration, and the direction, left or right, the tape head took when it left the cell. We call these the *previous symbol*, *previous state*, and *direction* of the cell, respectively.

We use $C = \{c_1, \ldots, c_k\} = \text{BCells} \cup \text{CCells} \cup \text{PCells}$ to denote the set of all configuration cells.

Configurations of M_w will be encoded by sequences of 2^n configuration cells from C. For such a sequence to correctly encode a configuration, we require that exactly one of its configuration cells $\alpha = (a, \tau)$ belongs to CCells. We also require the following:

- If the direction of τ is S (for stay), then there are no cells from PCells in the encoding.
- If the direction of τ is R (for right), then there is exactly one tile β from PCells in the sequence. It is placed to the *left* of α. The cell content of β, i.e., the current symbol it represents, is the *write symbol* of τ, its *previous symbol* is the read symbol of τ, its previous state is the from-state of τ and its direction is R. A symmetrical condition is imposed if the direction of τ is L.

The reason for this somewhat convoluted encoding is that it enables us to propagate information from one configuration encoding to the next. Let $Conf_1$ and $Conf_2$ be two sequences of 2^n configuration cells that correctly encode two configurations of M_w. We will argue how a few simple constraints can ensure that $Conf_2$ encodes a valid successor configuration of $Conf_1$. To this end, think of $Conf_2$ as lying on top of $Conf_1$ in the obvious manner (i.e., the leftmost configuration cell of $Conf_2$ lying on top of the leftmost configuration cell of Conf_1, etc.). We divide the set of constraints into two: a set of *horizontal* constraints ensuring consistency inside Conf_1 and inside Conf_2, and a set of *vertical* constraints ensuring consistency between Conf_1 and Conf_2. These constraints are similar to those used in *tiling games* [15]. Actually, the next part of our reduction constructs a special case of tiling games that is still 2EXPTIME-complete.

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

(H1) To the *left* of cells of the form $(a, \tau) \in$ CCells such that the direction of τ is R, there is always a cell $\beta \in$ PCells such that
- the direction of β is R,
- the current symbol of β is the write symol of τ,
- the previous symbol of β is the read symbol of τ,
- the previous state of β is the from-state of τ.

The converse also holds, i.e., such cells from CCells are the only ones allowed to the *right* of such cells from PCells.

(H2) To the *right* of cells of the form $(a, \tau) \in$ CCells such that the direction of τ is L, there is always a cell $\beta \in$ PCells such that
- the direction of β is L,
- the current symbol of β is the write symol of τ,
- the previous symbol of β is the read symbol of τ,
- the previous state of β is the from-state of τ.

The converse also holds, i.e., such cells from CCells are the only ones allowed to the *left* of such cells from PCells.

(H3) The only cell allowed to the *right* of a blank cell $\llcorner\!\lrcorner \in$ BCells is $\llcorner\!\lrcorner$.

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 $(b, \tau) \in$ CCells such that the direction of τ is either L or R and $b = a$.

(V2) On top of a cell $(a, \tau) \in$ CCells, the only allowed cells are
- any $\beta \in$ PCells such that the previous symbol of β is a and the previous state of β is the to-state of τ, and
- any $(b, \tau') \in$ CCells such that the from-state of τ' is the to-state of τ, the read letter of τ' is a, the write letter of τ' is b, and the direction of τ' is S.

(V3) On top of a cell $(a, (q, a', m)) \in$ PCells, the only allowed cells are
- $b \in$ BCells such that $b = a$, and
- any $(b, \tau) \in$ CCells such that $b = a$ and the direction of τ is L or R.

Condition (V1) encodes that M_w just moved to the current position from the left or from the right. The current position is not overwritten.

Figure 2 shows an example of a valid transition from C_1 to C_2 with respect to the horizontal and vertical constraints.

We now prove the following observation.

Observation 7 *Let $\mathit{Conf}_1 \in C^*$ encode a configuration κ_1 of M_w and let $\mathit{Conf}_2 \in C^*$. Then, the following are equivalent:*

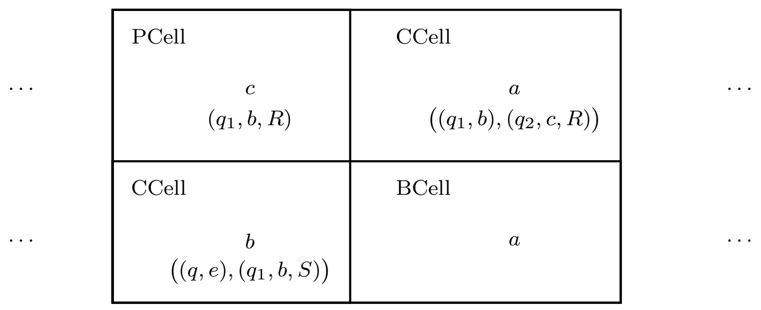


Fig. 2 A representation of a Turing Machine transition. The transition used is $\tau = ((q_1, b), (q_2, c, R))$, i.e., the machine is in state q_1, reads symbol b, writes a c, and moves to the right. The encoding of the cell where the head originally was (the upper left cell) "remembers" the previous state and tape symbol, so that the horizontal constraints can verify that the transition τ was actually allowed from the previous configuration.

– *$Conf_2$ has exactly one cell (a, τ) from CCells and both $H(M_w)$ and $V(M_w)$ are satisfied.*
– *$Conf_2$ encodes a configuration κ_2 of M_w such that $\kappa_1 \vdash_\tau \kappa_2$.*

Proof Let $Conf_1 = \alpha_1 \cdots \alpha_n$ and $Conf_2 = \beta_1 \cdots \beta_n$, where each $\alpha_i, \beta_i \in C$.

We first assume that all the constraints (H1–H3,V1–V3) hold and that $Conf_2$ has exactly one cell $(a, \tau) \in$ CCells, with $\tau = ((p_1, b), (p_2, c, m))$. Since $Conf_2$ has only one cell from CCells, we know from (H1) and (H2) that it also has at most one cell from PCells. For the remainder of the proof, we make a case distinction on whether a cell from PCells is present in $Conf_2$ or not.

Let us first consider the case where $Conf_2$ has no cell from PCells. Since $Conf_1$ encodes a configuration of M_w, we know that there is a unique $i \in \{1, \ldots, n\}$ such that $\alpha_i \in$ CCells. Since $Conf_2$ has no cell from PCells, we know from (V2) that $\beta_i = (a, \tau)$ is the unique cell from CCells in $Conf_2$. Let $(b_0, ((q_1, c_0), (q_2, a_0, m')))$ be the cell from CCells in $Conf_1$. From (V2) we get that $b_0 = b$, $q_2 = p_1$, $a_0 = a$, and $m = S$. This means that transition τ is possible from the configuration encoded by $Conf_1$ and that the result of applying it is to write an a and let the tape head stay where it is. In all positions except i we know that $Conf_1$ has a cell α_j from BCells $\cup$ PCells. Constraints (V1) and (V3) therefore imply that, on top of each such cell α_j is a cell β_j that represents the same tape symbol. Thus we can conclude that $Conf_2$ indeed represents a configuration that can be reached from the configuration encoded by $Conf_1$ by applying transition τ.

Next, we consider the case where $Conf_2$ has exactly one cell from PCells. Let $\alpha_i = (b_0, ((q_1, c_0), (q_2, a_0, m')))$ be the unique cell from CCells in $Conf_1$. The vertical constraints imply that cells from PCells are only allowed on top of cells from CCells. Thus β_i is the unique cell from PCells in $Conf_2$. Constraint (V2) implies that β_i is of the form $(y, (q_2, x), m')$, for some $y \in \Gamma$ and $m' \in \{L, R\}$. We assume that $m' = R$ (the other case is symmetrical).

By constraint (H1), the cell to the right of the cell from PCells in Conf_2 must belong to CCells. We know that this cell is $(a, ((p_1, b), (p_2, c, m)))$. By using (H1) we know that $y = c$, $x = b$, $q_2 = p_1$, and $m = m' = R$. This means that transition τ was possible from the configuration encoded by Conf_1 and that τ writes a c and moves the head to the right. Thus β_i in Conf_2 (the unique cell from PCells) represents the correct tape symbol $y = c$. By the same argument as above, all other cells represent the same tape symbols in Conf_2 as in Conf_1 and we can conclude that Conf_2 indeed represents a configuration that is reachable from the configuration encoded by Conf_1 by applying τ.

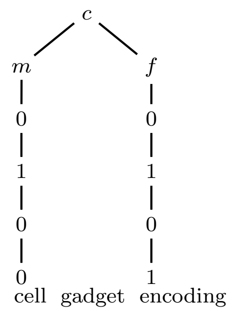
For the other direction, we assume that Conf_2 encodes that we arrived at κ_2 by $\tau = ((p_1, b), (p_2, c, m))$ and that $\kappa_1 \vdash_\tau \kappa_2$. It follows immediately that Conf_2 has exactly one cell from CCells and that this cell has the form (a, τ), for some $a \in \Gamma$. It is also immediate that the horizontal constraints (H1)–(H3) hold for Conf_2. We have to show that (V1)–(V3) are satisfied as well. Let i be the index of the tape cell the machine head was visiting in κ_1. Then all cells of Conf_2 other than i and (possibly) $i-1$ or $i+1$ will belong to BCells $\cup \{\llcorner\!\lrcorner\}$ and will represent the same tape symbol as the corresponding cells in Conf_1. Thus the vertical constraints hold for these cells.

If $m = S$, i.e., if τ is a stay transition, then the vertical constraints trivially hold for the cells at indices $i-1$ and $i+1$ as well and Conf_2 will have (a, τ) at index i, which satisfies all the vertical constraints. If $m = R$ the tape head will have moved right. In this case, Conf_2 has a cell from PCells at index i and a cell from CCells at index $i+1$. The definition of encoding a transition ensures that the cell at index i represents the tape symbol c, i.e., the symbol that is written by τ. The tape symbols at indices $i-1$ and $i+1$ remain unchanged, which ensures that all vertical constraints are satisfied. The case for $m = L$ is symmetrical. □

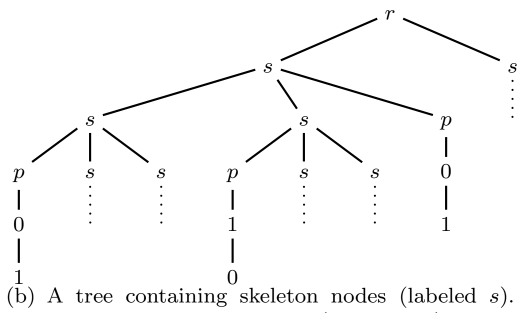
Encoding Configurations as Trees The most crucial part of the reduction is to use the query to detect when the transition relation of M_w is violated. To be able to do this, the query must be able to navigate from a node representing tape cell i in one configuration tree to the node representing cell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

Recall that $C = \{c_1, \ldots, c_k\}$ is the set of all distinct cell types we need to encode configurations of M_w. The size k of this set is polynomial in the size of M_w.

As we can assume without loss of generality that M_w never uses more than 2^n tape cells, we can encode configurations into the leaves of full binary trees of height n, where each leaf represents a configuration cell. For technical reasons, the configuration cells will not be represented by labels, but rather by *configuration 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.



(a) 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.



(b) A tree containing skeleton nodes (labeled s). Skeleton nodes that are the left (resp., right) s-child of their parent have a $p, 0, 1$ (resp., $p, 1, 0$) gadget.

Fig. 3 Gadgets for the CQ validity proof.

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; see Figure 3(b).

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.

Each leaf skeleton node (one that has no skeleton node children) is equipped with a *configuration cell gadget*. We describe the gadget for configuration cell c_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, where $k = |C|$. Here, the jth node from the top has label 0 if and only if c_i and c_j fulfill the vertical constraints (V1)–(V3), i.e, if c_j is allowed on top of c_i. Otherwise, it has label 1; see Figure 3(a). This concludes the description of an encoded configuration tree.

3.2.3 The Reduction

We now explain how to construct the NTA A_{CT} and the CQ Q_{CT} such that Q_{CT} is valid with respect to A_{CT} if and only if M_w does not have an accepting run.

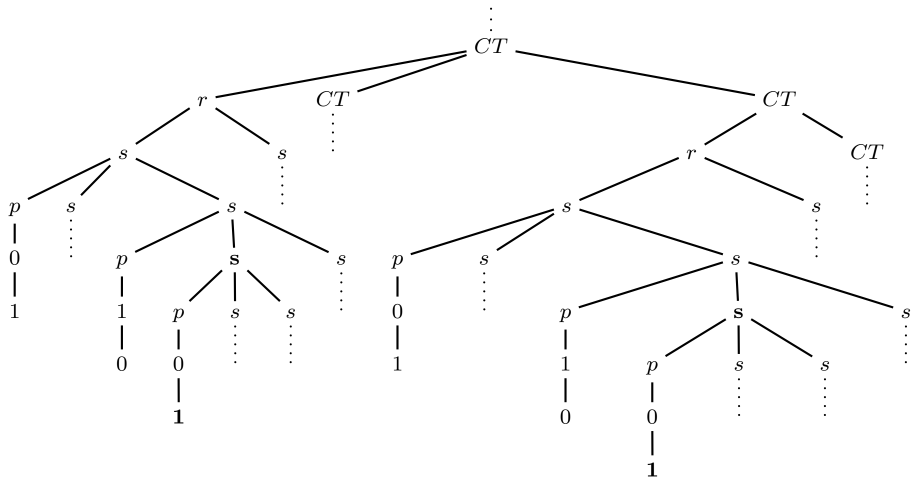


Fig. 4 An illustration of part of an encoded computation tree. Consider the two skeleton nodes labeled with boldface **s**. They are both at level $i = 3$ in their respective configuration trees. If, from the node with label one in the navigation gadget of the left boldface skeleton node (also in boldface), we go upwards $i + 4 = 7$ steps, we reach the root of the part of the tree depicted here. From the other boldface *1*, if we go $i + 5 = 8$ steps upward, we reach the same node. This would not have been the case if one of the boldface s nodes, but not the other, had been a right skeleton child of its parent.

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 a number of properties that we explain next. For technical reasons, we need t to start at the root with a path of length k, where k is again the number of distinct configuration cells corresponding to M_w, to the first CT-labeled node. All nodes on this path have label I and each of them has exactly one child. Further more, A_{CT} checks the following properties:

1. The subtree rooted at the highest CT-labeled node is an encoded computation 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 that is assured by the assumption that M_w is normalized. (Here, we are assuming that k is even, so that the path of I-labeled nodes above the highest CT-node has even length. If k is odd, the rules for odd and even depths are inversed.)

3. For each CT-labeled node representing a universal configuration, the two child CT-labeled nodes represent two encoded configuration trees with two *different* labels from CCells. This means that the two encoded successor configurations are different. Recall that M_w is normalized, so that transitions leaving universal configurations only change the machine state.
4. All horizontal constraints from $H(M_w)$ are satisfied in the encoded configurations.
5. The leftmost configuration cell of the highest encoded configuration tree is the start configuration cell $(\llcorner\lrcorner, ((q_0, \llcorner\lrcorner), (q_0, \llcorner\lrcorner, S)))$. 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, before 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. 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), 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.

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

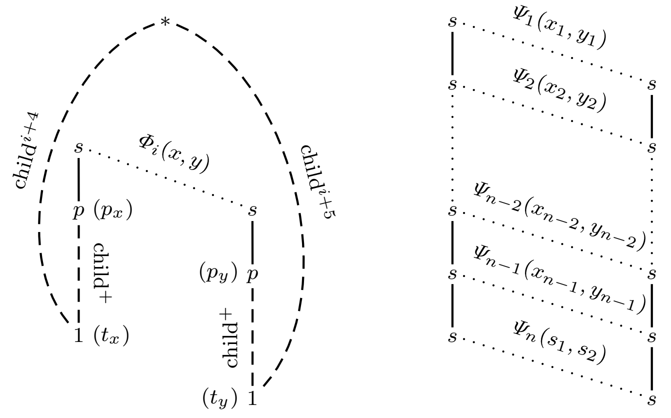


Fig. 5 Graphical representation of the queries $\Psi_i(x,y)$ and SameCell(s_1,s_2) from the proof of Theorem 6. The small labels in parentheses denote the variable names used in the proof.

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.

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

For a graphical representation of the subquery $\Psi_i(x,y)$ (and the subquery SameCell(s_1,s_2) defined below), see Figure 5.

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.

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

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 6. Intuitively, it will match a computation tree if it can find two successive configurations such that there is a position i where the cell in the second configuration is not allowed on top of the first configuration according to the vertical constraints. It does this by inspecting the cell gadgets representing the position in the two configurations. In the higher configuration, it looks for a 1, matched by query variable p_1, on the f-branch of the gadget, indicating that the corresponding cell is *not* allowed on top of the current one. It then verifies that the gadget in the second configuration has a 1, matched by p_2 at the same depth of its m-branch, indicating that it is an instance of the forbidden cell.

Summary This concludes the proof of Theorem 6. We have shown that given an EXPSPACE alternating Turing machine M and a word w, we can construct a nondeterministic tree automaton A_{CT} and a CQ(Child,Child^+) Q_{CT} in polynomial time, such that Q_{CT} is valid with respect to A_{CT} if and only if M has *no* accepting run on w. Since 2EXPTIME is closed under complement, this shows that CQ(Child,Child^+) validity (and thus also containment) with respect to an NTA is 2EXPTIME-hard.

3.2.4 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.

The main technical observation one has to make is stated in Lemma 8, which was probably first published in [43].

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 accepting runs of A. More formally, the annotated tree language of A is the set of trees t over $\mathrm{Alpha}(A) \times \mathrm{States}(A)$ where

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

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

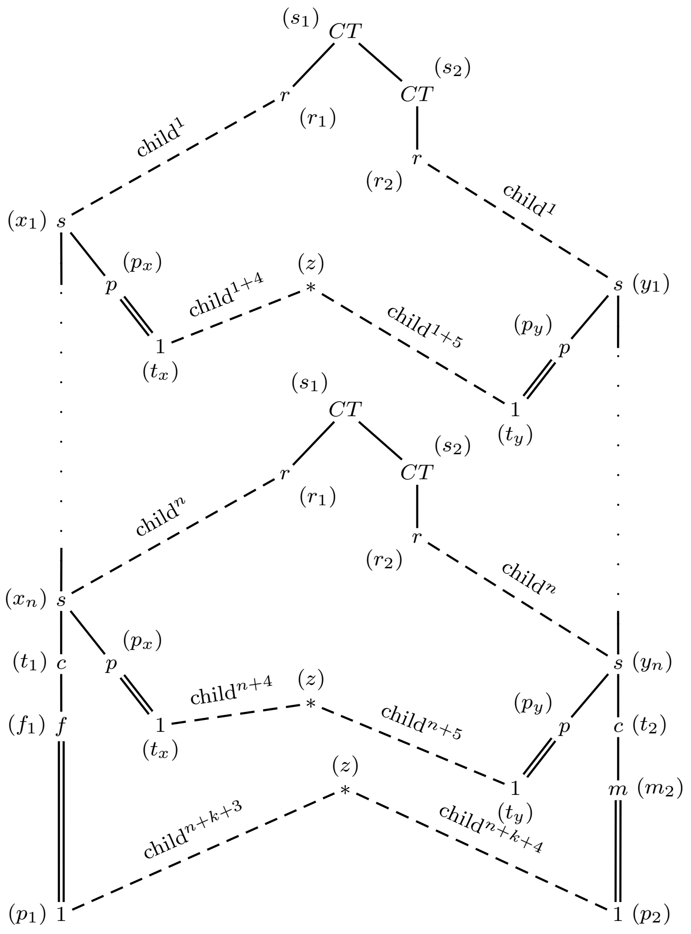


Fig. 6 Graphical representation of Q_{CT} from the proof of Theorem 6. The small labels in parentheses denote the variable names used in the proof.

Lemma 8 ([43]) *Given an NTA A, there exists a DTD D_A that recognizes the annotated tree language of A. Moreover, D_A can be constructed in quadratic time.*

Theorem 9 *Validity of CQ(Child,Child$^+$) with respect to a DTD is 2EXPTIME-complete.*

Proof 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 Alpha$(A_{CT}) \times$ 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 $a \in \{CT, r, s, p, 0, 1, c, m, f\}$ 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 $a \in \{CT, r, s, p, 1, m, f\}$) with $\mathit{Child}(x, y) \wedge a(y)$. The rest of the proof carries through. □

Theorem 9 also implies that validity with respect to a DTD becomes 2EXPTIME-complete for XPath patterns with the path intersection operator (as in XPath 2.0). We discuss this more precisely in Section 6.

4 Satisfiability

Satisfiability of CQs with respect to NTA has a drastically lower complexity than validity. In this section, we show that the problem is NP-complete. Further more, we show that the lower bound holds already for DTDs and CQs that only use one axis.

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. The small model is obtained by fairly standard cutting and pumping techniques (see also, e.g., [7, Lemma 1] or [27, Theorem 2, Theorem 3]). We start with the following lemma.

Lemma 10 *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 all 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 $T = \{\theta(x) \mid x \in \mathrm{Var}(Q)\}$, and let r be an accepting run of A on t. Furthermore, let S be the set of nodes that are lowest common ancestors of some subset of T of size at least 2.

Suppose that there exists a simple path ρ between two distinct vertices u and v in $T \cup S$ such that u is an ancestor of v, there are no nodes in $T \cup S$ on ρ, and the length of ρ is more than $|\mathrm{States}(A)| \cdot |\Sigma| + 1$. Notice that all descendants in T of nodes on ρ are also descendants of v. Towards a contradiction, assume that w is the lowermost node on ρ that has a descendant $w' \in T$ that is not a descendant of v. Then w would be the lowest common ancestor of v and w' and thus belong to S. In other words, no variable of Q is mapped by θ to any node in the subtrees that branch off from ρ.

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 $\mathrm{lab}^t(w) = \mathrm{lab}^t(w')$. 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 $1 + |\text{Var}(Q)| \cdot (|\Sigma| \cdot |\text{States}(A)| + 1)$. $\square$

Lemma 10 gives us the main machinery to prove the general NP upper bound on satisfiability:

Theorem 11 *Satisfiability of CQs with respect to an NTA is in NP.*

Proof We can assume w.l.o.g. that the NTA A is *reduced*, i.e., each state of A can be used in an accepting run.[3] We know from Lemma 10 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 all $x, y \in \text{Var}(Q)$, the distance between $\theta(x)$ and $\theta(y)$ is small (polynomial). In general, t can be exponentially large. If Q is satisfiable with respect to A, however, 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 $z_0, \ldots, z_n \in \text{States}(A)^*$ 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. $\square$

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 some axes, the result is already known:[4]

Theorem 12 (Wood [47]) *Let Axis be any element of* $\{$*Child, Child*$^+$, *Child*$^*\}$. *Then Satisfiability for CQs using only the relation Axis w.r.t. a DTD is NP-hard.*

[3] Transforming an NTA to a reduced NTA can be done in polynomial time by first performing an emptiness test for every state of A, followed by a reachability test. Section 4.2 of [35] describes an algorithm for reducing a *DTD*. The algorithm for NTAs is analogous.

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

The proof relies on the following Lemma:

Lemma 13 (Wood [47]) *The following problem is NP-hard. Given a 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 one of u and v belongs to V'. Let $G = (V, E)$ and k be an arbitrary instance of Vertex Cover. We will construct a 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 $E_i = \{e_{i,1}, \ldots, e_{i,m_i}\} \subseteq E$ be the set 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. For each $1 \leq i \leq n$, let s_i be the string $e_{i,1} \cdots e_{i,m_i}$. Let S be the regular expression

$$s_1 + \cdots + s_n.$$

Then $R = (S\#)^{k-1}S$, that is, k concatenated occurrences of expression S, separated by #-symbols. This means that every word in $L(R)$ is a concatenation of k strings of edge symbols, separated by #, where each string of edge symbols represent the edges incident to some vertex.

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 concatenating $s_{j_1}, \ldots, s_{j_k}$, separated by #. 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. The 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 $L(S)$. Then each w_i is equal to s_{j_i} for some $1 \leq j_i \leq n$. Since w contains all symbols in Σ, the set $V' = \{v_{j_1}, \ldots, v_{j_k}\}$ is a vertex cover for G and $|V'| \leq k$. $\square$

Remark 14 Wood's Lemma [47] already holds if the regular expression is *deterministic* (sometimes also called *one-unambiguous* [12]). Intuitively, a regular expression is deterministic if, when reading a word from left to right without looking ahead, it is always clear where in the expression the next symbol can be matched. In DTDs in practice, regular expressions must always be deterministic.

Formally, deterministic regular expressions are defined as follows [12]. Let $\bar{r}$ stand for the RE obtained from r by replacing, for every integer i and alphabet symbol a, the i-th occurrence of a in r by a_i (counting occurrences from left to right). For example, for $r = b^*a(b^*a)^*$ we have $\bar{r} = b_1^*a_1(b_2^*a_2)^*$. A regular expression r is *deterministic* (or *one-unambiguous*) if there are no words wa_iv and wa_jv' in $L(\bar{r})$ such that $a \in \Sigma$ and $i \neq j$. For instance, the expression $(a + b)^*a$ is not deterministic since both words a_2 and a_1a_2 are in $L((a_1+b_1)^*a_2)$. The equivalent expression $b^*a(b^*a)^*$ is deterministic. Not every regular expression can be determinized, that is, converted to an equivalent deterministic regular expression. In fact, deciding if a given expression can be determinized is PSPACE-complete [17,33].

Wood [47] remarks that 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.

It is not hard to prove Theorem 12 given Lemma 13:

Proof (of Theorem 12) By reduction from the problem in Lemma 13. 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 CQ tests whether each Σ-symbol occurs below the root. □

Notice that Wood's proof already shows hardness for CQs for which the underlying graph is a tree or, even stronger, star-shaped. Indeed, Wood's original result is on XPath queries, which are tree-shaped by design.

For the remaining cases, we will reduce from the Shortest Common Supersequence problem; or the Shortest Common Superstring problem, both of which are known to be NP-complete [42,25]. We say that 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. 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.

Theorem 15 *Let Axis be an any element of {Child, Child$^+$, Child*, NextSibling, NextSibling$^+$, NextSibling*, Following}. Then, testing Satisfiability for CQ(Axis) w.r.t. a DTD is NP-hard.*

Proof Three cases are immediate from Theorem 12. 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. To this end, 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

$$\begin{tikzcd}
b_1^1 \arrow[r] & b_1^2 \arrow[r] & \cdots \arrow[r] & b_1^{n_1} \\
\vdots & & & \vdots \\
b_m^1 \arrow[r] & b_m^2 \arrow[r] & \cdots \arrow[r] & b_m^{n_m}
\end{tikzcd}$$

Fig. 7 Query for the proof of Theorem 15.

the query Q is defined as shown in Figure 7. 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*).

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

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

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

5 Queries with Data Values

A *data tree* is a tree in which each node u carries, besides its label lab(u), a *data value* from a countably infinite data domain Δ (see also [11]).[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: CQ($\sim$), CQ($\not\sim$), and 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 $Q \in$ 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 2.2, but with "tree" replaced by "data tree".

Our problems of interest for queries with data values are the same problems as defined in Section 2.3, 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 16 *Satisfiability of CQs($\sim, \not\sim$) w.r.t. an NTA is NP-complete.*

The lower bound just follows from Theorem 15. For the upper bound, the proof of Lemma 10 and Theorem 11 straightforwardly carries over to data trees.

The lower bound of the following theorem follows from Theorem 6. The upper bound follows from Theorem 20, which subsumes it.

Theorem 17 *Validity of CQ($\sim, \not\sim$) w.r.t. an NTA is 2EXPTIME-complete.*

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

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

Next, we consider containment w.r.t. a schema. We write

$$\textsc{Containment}(X|Y)$$

for the problem of determining whether $L(P) \cap L(A) \subseteq L(Q)$ for a query $P \in$ CQ(X), a query $Q \in$ CQ(Y) and an NTA A. For instance, CONTAINMENT($\sim$ | $\sim, \not\sim$) is about containment of queries with data equalities in queries with data equalities and inequalities.

As can be seen in Table 1, 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 both equalities and inequalities. In the latter case the problem is undecidable (Theorem 25).

5.1 Complexity Upper Bounds

The proofs of the upper bounds make use of transformations from certain non-data trees to data trees. We define these transformations next.

We assume that Δ contains pairwise distinct values $d_0, d_1, d_2, \ldots$. Given a finite alphabet Σ and $n \in \mathbb{N}$, let Σ_n, denote the alphabet $\Sigma \times \{d_1, \ldots, d_n, *\}$. The set of (non-data) trees over Σ_n is denoted $T(\Sigma_n)$.

We define functions $f_{\not\sim}$ and $f_{\sim}$, mapping trees from $T(\Sigma_n)$ to data trees. They leave the sets of vertices and edges unchanged. In a nutshell, both functions map nodes with label (a, d_i) to nodes with label a and data value d_i. They differ in how nodes with labels $(a, *)$ are handled: $f_{\sim}$ maps all these nodes to nodes with label a and the *same* data value d_0, whereas $f_{\not\sim}$ maps all those nodes to nodes with *different* data values. More formally, $f_{\sim}$ and $f_{\not\sim}$ fulfil the following conditions, for every tree $t \in T(\Sigma_n)$, for some $d_0 \notin \{d_1, \ldots, d_n\}$.

1. If $\mathrm{lab}^t(v) = (a, d_i)$, for $a \in \Sigma$ and $i \in \{1, \ldots, n\}$, then node v of $f_{\sim}(t)$ and $f_{\not\sim}(t)$ have label a and data value d_i.
2. If $\mathrm{lab}^t(v) = (a, *)$, for $a \in \Sigma$, then node v of $f_{\sim}(t)$ has label a and data value d_0.
3. If $\mathrm{lab}^t(v) = (a, *)$, for $a \in \Sigma$, then node v of $f_{\not\sim}(t)$ has label a and a data value that does not appear elsewhere in the tree.

Lemma 18 *Given a query Q in CQ($\sim, \not\sim$) with n variables, one can construct NTAs $A_Q^{\sim}$ and $A_Q^{\not\sim}$ in exponential time such that for each Σ_n-tree t it holds that*

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

(a) $t \in L(A_Q^{\sim})$ *if and only if* $f_{\sim}(t) \in L(Q)$*, and*

(b) $t \in L(A_Q^{\not\sim})$ *if and only if* $f_{\not\sim}(t) \in L(Q)$*.*

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

Given $Q \in CQ(\sim, \not\sim)$, we construct an NTA A over Σ_n such that $t \in L(A)$ if and only if $f_{\not\sim}(t) \in L(Q)$. A state of A has the form (X_a, X_c, X_d, F), where X_a, X_c, 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, $(X_a, X_c, X_d, F) \in \mathrm{States}(A)$ if (X_a, X_c, X_d) fulfill the conditions (S1)–(S4) in the proof of Lemma 4 and additionally

(S5) 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_c, or none of them do, and
(S6) 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_c.

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

Rules(A): contains all rules of the form

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

where (R1) and (R2) from the proof of Lemma 4 are satisfied, and
(R3) for each $x \in X_c$, we have $F(x) = \lambda$;
(R4) for each $(X_a^1, X_c^1, X_d^1, F^1) \cdots (X_a^m, X_c^m, X_d^m, F^m) \in L(R)$, we have $F^1 = \cdots = F^m = F$.

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 by a factor of at most $|\mathrm{Var}(Q)|^n$. Clearly, thanks to (R4), in each accepting run of A, all nodes have the same evaluation function F. We can therefore consider this function as independent of a particular node of t.

We show that $L(A) = \{t \in T(\Sigma_n) \mid f_{\not\sim}(t) \models Q\}$. For the inclusion from left to right, let $t \in L(A)$ with some accepting run r. We define the valuation θ of Q by letting $\theta(x)$ be the unique node v satisfying $r(v) = (X_a, X_c, X_d, F)$ with $x \in X_c$, for some X_a, X_c, X_d, F. The proof of Lemma 4 immediately implies that with this valuation $f_{\not\sim}(t)$ satisfies all atoms of Q not involving $\sim$. We claim that $f_{\not\sim}(t)$ and θ also satisfy all $\sim$-atoms of Q.

Indeed, if $x \sim y$ is an atom of Q, then, thanks to (S5), $F(x) = F(y)$. Thus, $x \sim y$ is satisfied if $F(x) \in \{d_1, \ldots, d_n\}$. If $F(x) = *$, again thanks to (S5), there must be a node v on which $\{x, y\} \subseteq X_c$ holds, implying that $x \sim y$ holds.

For atoms $x \not\sim y$ we consider two cases: if $F(x) = F(y) = *$, (S6) guarantees that $\theta(x) \neq \theta(y)$ and by the definition of $f_{\not\sim}(t)$ the two nodes have different data values. Otherwise, (S6) guarantees that $F(x) \neq F(y)$ and again $x \not\sim y$ is satisfied. Hence, $f_{\not\sim}(t) \models Q$.

For the inclusion from right to left, let Q' be the query resulting from Q by removing all (positive and negative) $\sim$-atoms. Let A' be the NTA for Q' as guaranteed by Lemma 4.

Let $t \in T(\Sigma_n)$ such that $f_{\not\sim}(t) \models Q$ and let θ be a satisfaction of Q on $f_{\not\sim}(t)$. We define an evaluation function F by letting $F(x)$ be the second component of the label of $\theta(x)$ in t, for each $x \in \text{Var}(Q)$.

Let t' be the Σ-tree obtained from t by projecting all labels to Σ. It is easy to see that $t' \models Q'$. Let r' be an accepting run of A' on t' corresponding to θ, as it was established in the proof of Lemma 4.

Let r be obtained from r' by adding F as fourth component to each state $r'(v)$ for nodes v of t. We claim that r is a run of A.

First of all, we need to show that r only needs states that fulfil (S5) and (S6). The first statements of both conditions are immediately guaranteed by the definition of F and the fact that θ is a satisfaction of Q. The second statement of (S5) holds as well: the satisfaction of each atom $x \sim y$ by θ and the definition of $f_{\not\sim}(t)$ guarantee that, in case $F(x) = F(y) = *$, we have $\theta(x) = \theta(y)$, and therefore (S5) holds. Similarly for the second statment of (S6).

In r' it holds that, for each variable x, there exists exactly one node v with a state (X_a, X_c, X_d, F) with $x \in X_c$ and $v = \theta(x)$. From this (R3) immediately follows. Condition (R4) is guaranteed by the construction of r. $\square$

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

Let P be a query with variables from $\{x_1, \dots, x_n\}$ 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 by assigning a label to every node v as follows. If $\text{lab}^{t_d}(v) = a$,

- then v gets label (a, d_j), if $j \in \{1, \dots, n\}$ is minimal with $\theta(x_j) \sim v$, or
- v gets label $(a, *)$ if no such j exists.

Lemma 19 *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 be as stated. It is straightforward that θ is a satisfaction for P on both $f_{\sim}(g_n(t_d, \theta))$ and $f_{\not\sim}(g_n(t_d, \theta))$.

To show (b), let us assume that $Q \in \text{CQ}(\sim)$ and $f_{\not\sim}(g_n(t_d, \theta)) \models Q$. Since 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, we can conclude that $t_d \models Q$, a contradiction. Therefore, $f_{\not\sim}(g_n(t_d, \theta)) \not\models Q$.

The proof of (c) similarly uses the fact that $\sim$ on t_d is a refinement of $\sim$ on $f_{\sim}(g_n(t_d, \theta))$. $\square$

We are now ready to state and prove our upper bound results.

Theorem 20 *Each of* Containment($\sim, \not\sim \mid \sim$)*,* Containment($\sim, \not\sim \mid \not\sim$)*,* Containment($\sim \mid \sim, \not\sim$)*, w.r.t. an NTA is 2EXPTIME-complete.*

Proof Hardness is immediate from Theorem 6.

For the upper bound on Containment($\sim, \not\sim \mid \sim$) we observe that from Lemma 19 (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 18 with the NTA A_n which accepts a Σ_n-tree t if its Σ-projection is accepted by A. The former two automata are of at most exponential size, their deterministic counterparts are of at most doubly exponential size and therefore it can be tested in doubly exponential time whether $L(A_P^{\not\sim}) \cap L(A_n) \subseteq L(A_Q^{\not\sim})$.

The upper bound on Containment($\sim, \not\sim \mid \not\sim$) follows similarly from Lemma 19 (a) and (c).

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

Hence, $\sim$ and $\not\sim$ do not 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 $\not\sim$.

5.2 Undecidability Results

We now turn to the proof of undecidability of Containment($\not\sim \mid \sim, \not\sim$) (and thus also: Containment($\sim, \not\sim \mid \sim, \not\sim$)). We first prove that validity with respect to an NTA is undecidable for UCQ($\sim, \not\sim$) and show how to adapt that to a reduction to Containment($\not\sim \mid \sim, \not\sim$) later on.

Theorem 21 *Validity of UCQ($\sim, \not\sim$) queries w.r.t. NTAs is undecidable.*

Proof Our proof, inspired by a proof from [40], is by a reduction from *Post's Correspondence Problem* (PCP). In the proofs of Theorems 23 and 25 below, this reduction will be adapted for the respective settings. Some choices in the presentation of the reduction were made with these adaptations in mind. Readers should thus not be surprised if we do not always choose the most obvious option to express properties of trees.

An *instance* of PCP over alphabet $\Gamma = \{a, b\}$ is a sequence $(w_1, u_1), \ldots, (w_n, u_n)$ of pairs, where $w_i, u_i \in \Gamma^+$, for $i \in \{1, \ldots, n\}$. A *solution* to an instance is a non-empty sequence $i_1, \ldots, i_m \in \{1, \ldots, n\}$ such that $w_{i_1} \ldots w_{i_m} = u_{i_1} \ldots u_{i_m}$. It is known that the set of PCP instances for which a solution exists is undecidable [41].

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

The set Σ of labels to be used by A and Q is defined as $\{r, \#\} \uplus I \uplus \Gamma$, where

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

In this reduction, solution candidates will be encoded over unary trees. 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 *words*, i.e., words 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 words to reason about these unary trees. The queries we construct will be stated as tree queries, but can be read as queries over words by interpreting *Child* as the next position predicate and *Child*$^+$, *Child**as the transitive and the transitive and reflexive closure of *Child*, respectively.

For a data word w and a label set X, let $w_{|X}$ be the word over X obtained from w by removing all data values and deleting all positions with labels not in X.

If R has a solution, then there is a word $rw\#u\#$ that is accepted by A such that $w_{|\Gamma} = u_{|\Gamma}$ and $w_{|I} = u_{|I}$.

The intuition behind our proof is as follows. We encode solution candidates for R by data words $rw\#u\#$ such that the following conditions hold.

(ENC1): No data value appears more than twice below the root.
(ENC2): The occurence of r and the two occurrences of $\#$ have the same data value.
(ENC3): If two positions are at corresponding positions in $w_{|I}$ and $u_{|I}$, they have the same data value.
(ENC4): If two positions are at corresponding positions in $w_{|\Gamma}$ and $u_{|\Gamma}$, they have the same data value.
(ENC5): The length of $w_{|I}$ equals the length of $u_{|I}$ and the length of $w_{|\Gamma}$ equals the length of $u_{|\Gamma}$.
(ENC6): If two positions have the same data value, they carry the same label, unless one carries r and the other $\#$.

If a data word satisfies the requirements (ENC1)–(ENC6), we say that it is a *good* encoding. We construct Q such that it matches every word accepted by A that is not a good encoding. We further show that R has a solution if and only if there exists a good encoding that is accepted by A. Hence, Q is valid w.r.t. A if and only if R has no solution.

We are next going to define the subqueries of query Q. We can express that (ENC1) or (ENC2) is violated by

$$\begin{aligned} Q_1 &\equiv \exists u, x, y, z : r(u) \wedge \mathit{Child}^+(u,x) \wedge \mathit{Child}^+(x,y) \\ &\qquad\qquad \wedge \mathit{Child}^+(y,z) \wedge u \sim x \wedge x \sim y \wedge y \sim z \\ Q_{2,r} &\equiv \exists x, y : r(x) \wedge \#(y) \wedge \mathit{Child}^+(x,y) \wedge x \not\sim y \\ Q_{2,\#} &\equiv \exists x, y : \#(x) \wedge \#(y) \wedge \mathit{Child}^+(x,y) \wedge x \not\sim y \end{aligned}$$

For the remaining conditions we use a binary auxiliary query Separated(x,y) that expresses that x is in the first half of the word and y in the second.[6]

$$\begin{aligned} \text{Separated}(x,y) \equiv \exists x', x'', y' \ &: \ \#(x') \wedge \#(x'') \wedge \#(y') \\ &\wedge \mathit{Child}^+(x,x') \wedge \mathit{Child}^+(x',x'') \wedge \mathit{Child}^+(y',y) \end{aligned}$$

We next define a query $Q_{w,u}$ parameterized by two words w, u over $\Sigma \cup \{*\}$ which will be used to express violations of conditions (ENC3) and (ENC4). In a nutshell, $Q_{w,u}$ expresses that there are positions x_1 and y_1 of the first and second half, respectively, that have the same data value, the subwords starting at x_1 and y_1 match w and u, but their end-positions have different data values. Queries of this form will catch many data words that fail to be good encodings.

Let $v = v_1 \cdots v_k$ and $z = z_1 \cdots z_\ell$ be words over $\Sigma \cup \{*\}$ for some $k, \ell \geq 2$. Then we define

$$\begin{aligned} Q_{v,z} \equiv \exists x_1, \ldots, x_k, y_1, \ldots, y_\ell &: \text{Separated}(x_1, y_1) \wedge x_1 \sim y_1 \wedge x_k \not\sim y_\ell \\ \wedge \bigwedge_{i=1}^{k-1} \mathit{Child}(x_i, x_{i+1}) \wedge \bigwedge_{i=1}^{k} v_i(x_i) &\wedge \bigwedge_{i=1}^{\ell-1} \mathit{Child}(y_i, y_{i+1}) \wedge \bigwedge_{i=1}^{\ell} z_i(y_i) \end{aligned}$$

Here, $*(x)$ has to be interpreted as TRUE.

We can express that condition (ENC3) is violated by the disjunction of the following queries.

– $Q_{r*,\#*}$, expressing that the first I-position after the root and the first I-position after the first $\#$, respectively, have different data values.
– $Q_{I_i w_i *, I_i u_i *}$, for each $i \in \{1, \ldots, n\}$, expressing that the I-pair following some I-pair with equal data values has different data values.

We note that A makes sure that in all cases the two $*$-positions carry labels from I. Notice that these two queries have size polynomial in the PCP instance.

Violations of condition (ENC4) can be expressed similarly, but there are more cases to distinguish due to the possible I-symbols between two Γ-symbols. We use the following queries.

[6] This definition is done with the proof of Theorem 23 in the back of our minds and therefore more complicated than a reader might have expected. In this proof, the reader should think of x' and y' as being mapped to the same node.

- $Q_{r**,\#**}$, expressing that the first Γ-position after the root and the first Γ-position after the first $\#$-position, have different data values.
- $Q_{\sigma_1\sigma_2,\tau_1\tau_2}$ for each combination of $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \Gamma$ expressing that a Γ-pair with identical data values is immediately followed by a Γ-pair with different data values.
- $Q_{\sigma_1\sigma\sigma_2,\tau_1\tau_2}$ for each combination of $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \Gamma$ and $\sigma \in I$, expressing that a Γ-pair with identical data values is followed by a Γ-pair with different data values, but there is an intermediate I-position in the v-word.
- $Q_{\sigma_1\sigma_2,\tau_1\tau\tau_2}$ for each combination of $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \Gamma$ and $\tau \in I$, analogously, with an intermediate I-position in the z-word.
- $Q_{\sigma_1\sigma\sigma_2,\tau_1\tau\tau_2}$ for each combination of $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \Gamma$ and $\sigma, \tau \in I$, analogously, with intermediate I-positions in both words.

As will be detailed below, (ENC5) does not require an extra query.
Finally, violations of (ENC6) are expressed by the queries

$$Q_{\sigma:\tau} \equiv \exists x, y : \sigma(x) \wedge \tau(y) \wedge x \sim y,$$

for every pair $(\sigma, \tau) \in \Sigma^2$ with $\sigma \neq \tau$ and $(\sigma, \tau) \notin \{(r, \#), (\#, r)\}$.

Query Q is the disjunction of all the CQs for (ENC1)–(ENC6) defined above.

We now 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 word with label sequence $rw\#u\#$ such that (ENC1)–(ENC6) hold for s. Clearly, s is accepted by A. It is easy to verify that none of the disjuncts of Q is satisfied by s. Thus we can conclude that $s \in L(A) - L(Q)$.

For the only-if-direction, assume that data word 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 words. Furthermore, failure of Q_1 and Q_2 ensure (ENC1) and (ENC2).

Since neither $Q_{r*,\#*}$ nor any of the $Q_{I_i w_i *, I_i u_i *}$ queries match s, we can conclude that the first two I-positions have the same data value and, whenever two I-positions have the same data value, the next two I-positions have the same data values as well. We note that thanks to A, the pair following an I-pair needs to be an I-pair or the pair of $\#$-positions. Altogether, (ENC3) holds and $|w_{|I}| = |u_{|I}|$. (ENC4) and $|w_{|\Gamma}| = |u_{|\Gamma}|$ follow in the same fashion. Thus, (ENC5) holds as well. Finally, (ENC6) holds, since none of the formulas $Q_{\sigma:\tau}$ matches s.

We can conclude that $w_{|I} = u_{|I}$ and $w_{|\Gamma} = u_{|\Sigma}$ and that thus R has a solution. □

By some extra work and a different way of encoding solution candidates, the proof of Theorem 21 can be extended to show that Containment($\not\sim$ | $\sim, \not\sim$) is undecidable. To this end, we need the following lemma.

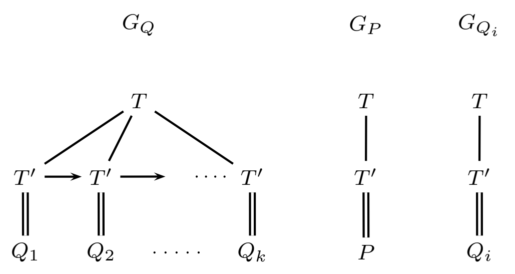


Fig. 8 Gadgets used in the proof of Lemma 22.

Lemma 22 *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 such that $L(P) \cap L(A) \subseteq L(Q_1) \cup \cdots \cup L(Q_k)$ if and only if $L(P') \cap L(A') \subseteq L(Q')$.*

Proof This proof is an adaptation of a proof from [38]. 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 $P \subseteq Q_1 \cup \cdots \cup Q_k$ w.r.t. A. We assume that all input queries are satisfiable, even with respect to A, which can be tested in NP. Furthermore, all queries should have pairwise disjoint variable sets.

The main idea is in Figure 9. Query P is inside the G_P-gadget in P' and the queries Q_i are inside the G_{Q_i}-gadgets in Q'. Intuitively, P' and Q' work together as follows. Imagine that a tree t matches P' and has the sequence of $2k-1$ S-nodes as indicated in P'. The middle S-node in this sequence is attached to a subtree in $L(G_P)$. Now, in order for Q' to also match t, one of the (k many) S-nodes in Q' must match the middle S-node in t, which means that one of the G_{Q_i} matches the tree in $L(G_P)$.

We now discuss the construction in detail. Figure 8 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 there is a variable x such that the query contains the atom $T'(x)$ and, for *every* variable y in the copy of Q_1, the atom $\mathit{Child}^+(x, y)$. The arrows between the T'-labeled nodes in G_Q indicate *NextSibling* predicates. The gadget G_{Q_i} is parameterized by i. It is crucial but easy to see that, for every i, $G_Q \subseteq G_{Q_i}$. Figure 9 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.
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.

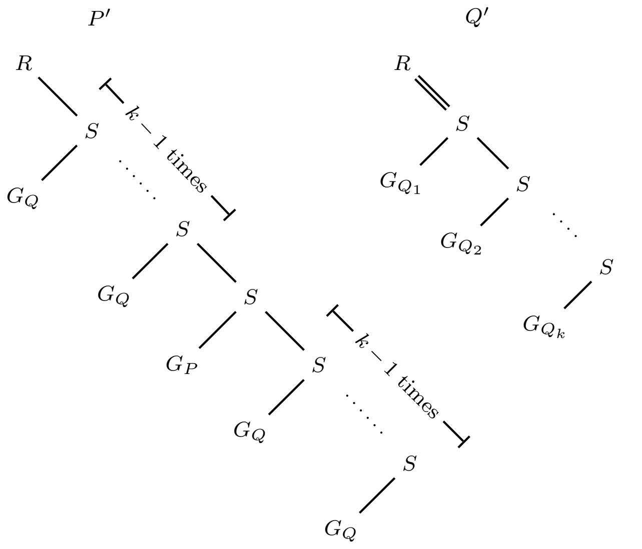


Fig. 9 Queries P' and Q' from the proof of Lemma 22.

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 $P \subseteq Q_1 \cup \cdots \cup Q_k$ 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 $Q_1 \cup \cdots \cup Q_k$ 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 $P \not\subseteq Q_1 \cup \cdots \cup Q_k$ w.r.t. A. Let p be a tree in $(L(P) \cap L(A)) - L(Q_1 \cup \cdots \cup Q_k)$. Let t be a tree in $L(P') \cap L(A')$, whose existence is guaranteed by our assumption, and define t_k and t'_k as above.Replace t'_k by p in t. The resulting tree t_p still belongs to $L(P') \cap L(A')$,

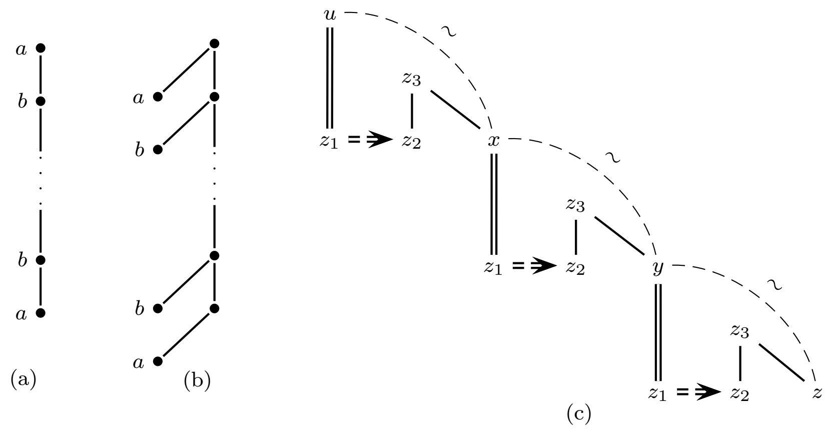


Fig. 10 Trees and queries used in the proof of Theorem 23.

since p is accepted by A and P matches in p. But since no Q_i, for $i \in \{1, \dots, k\}$ matches in p, there is no matching of Q' in t_p. Thus $P' \not\subseteq Q'$ w.r.t. A'. □

Theorem 23 CONTAINMENT($\not\sim$ | $\sim, \not\sim$) *is undecidable.*

Proof It would be tempting to conclude Theorem 23 directly from Theorem 21 and Lemma 22, since Theorem 23 shows undecidability of validity, that is, basically, query containment with TRUE (and thus a query without $\sim$) as the left-hand query. However, the reduction in the proof of Lemma 22 might introduce $\sim$-atoms in P' through the pattern G_Q, which has to fulfill $G_Q \subseteq G_{Q_i}$, for every i.

Therefore, we modify the proof of Theorem 21, apply Lemma 22, and do some final adaptations. Given an instance R of PCP, we first construct a query Q that is a disjunction of CQs and an automaton A such that Q is valid w.r.t. A, if and only if R has no solution. To remove the disjunction in Q, we then use a modification of the construction in the proof of Lemma 22.

Altogether, we construct queries $P_{\not\sim} \in \mathrm{CQ}(\not\sim)$ and $Q_{\sim,\not\sim} \in \mathrm{CQ}(\sim, \not\sim)$, and NTA A' such that Q is valid w.r.t. A if and only if $L(P_{\not\sim}) \cap L(A') \subseteq L(Q_{\sim,\not\sim})$.

We first describe how we modify the encoding of solution candidates for R from the proof of Theorem 21. We modify the unary trees used in that proof as follows. Each node gets a new extra leftmost child. The new nodes inherit their label from their parent node. The "old" nodes all get a new "blank" label (see Figure 10(a) and 10(b)).

Clearly, the automaton A can be adapted to take care of this new shape of trees. The disjuncts Q_i of the query Q are transformed into queries Q'_i that reflect the change of the encoding as follows. Each atomic formula $a(x)$, for $a \in \Sigma$ is replaced by $\exists x'\, \mathit{Child}(x, x') \wedge a(x')$, where x' is a fresh variable. At the same time, for each original variable z of Q_i a new atom blank(z) is added, ensuring that these variables can only be matched by ("original") nodes on the backbone of the tree. It is easy to see that a formula Q_i holds on an "old"

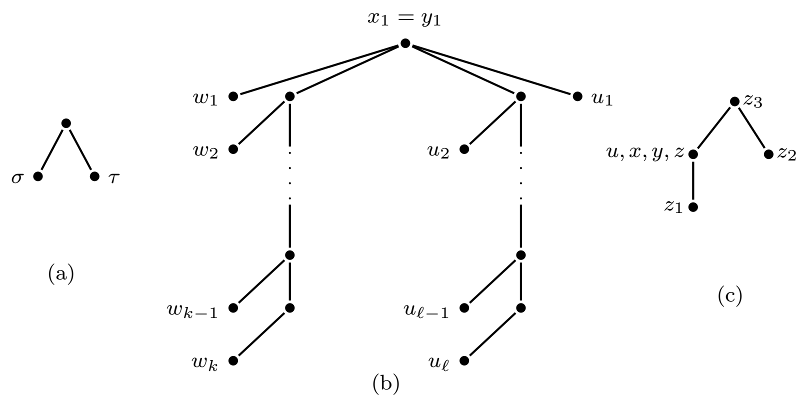


Fig. 11 Data trees witnessing the satisfiability of all queries $Q_j^{\not\sim}$ in the proof of Theorem 23.

encoding of a solution candidate if and only if Q_i' holds on its "new" encoding. The first query Q_1 (corresponding to (ENC1)) is modified even further, by replacing $\mathit{Child}^+(u,x)$ with

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

and $\mathit{Child}^+(x,y)$ and $\mathit{Child}^+(y,z)$ by the same kind of gadget.

The resulting query Q_1' is depicted in Figure 10(c). We note that the latter modification does not change the semantics on the intended trees, resulting from a solution candidate by the above "new" encoding.

The NTA A' and the query $Q_{\sim,\not\sim} = Q'$ are obtained from the adapted automaton A and the disjunction of the queries Q_i' just as in the proof of Lemma 22.

We still need to define $P_\sim$. To this end, let P' be the query that would be obtained from applying the proof of Lemma 22 to $Q_1' \cup \dots \cup Q_m'$ and A. From the construction of Lemma 22, we have that P' contains several occurrences of the gadget $G_{Q'}$, which can contain data equalities. We change $G_{Q'}$ to $G_{Q'}^{\not\sim} \in \mathrm{CQ}(\not\sim)$ such that, for each $i \leq m$, we have $G_{Q'}^{\not\sim} \subseteq Q_i'$.To this end, we replace subqueries Q_j' in $G_{Q'}$ by subqueries $Q_j^{\not\sim}$, in which every atom $x \sim y$ is replaced[7] by $x = y$. Clearly, $Q_j^{\not\sim} \subseteq Q_j'$ holds, for every j. However, to mimic the proof of Lemma 22, we need to make sure that $G_{Q'}$, and therefore each of the new queries $Q_j^{\not\sim}$, is satisfiable by some data tree (not necessarily of the shape of encodings of solution candidates).

For queries of the form $Q_{\sigma:\tau}$ this is actually very easy: $Q_{\sigma:\tau}'$ is the following query (in prenex form and after identification of y and x):

$$\exists x, x_1, x_2 : \mathit{Child}(x,x_1) \wedge \sigma(x_1) \wedge \mathit{Child}(x,x_2) \wedge \tau(x_2),$$

[7] Of course, the resulting equality atoms can be removed by suitable variable renaming.

which is satisfiable as Figure 11(a) illustrates. Queries $Q_j^{\not\sim}$ resulting from formulas of the form $Q_{w,u}$ are satisfiable as well, as the reader may conclude from Figure 11(b).

Since neither $Q_{2,r}$ nor $Q_{2,\#}$ do contain any $\sim$-atoms, the only remaining, but also the most complicated, case is query $Q_1^{\not\sim}$. However, as Figure 11(c) illustrates, Q_1' is also satisfiable by mapping its variables as indicated.

Altogether, we have described a reduction from PCP to CONTAINMENT($\not\sim$ $\mid \sim, \not\sim$) and we can conclude that the latter is undecidable. □

5.3 Containment without schema information

It actually turns out that if both queries can use $\sim$ and $\not\sim$, the schema automaton from Theorem 23 can be eliminated. Thus containment for CQ($\sim, \not\sim$) queries is undecidable even without schema information.

For the proof, we first show a counterpart of Lemma 22.

Lemma 24 *The problem to determine whether $L(P) \subseteq L(Q_1) \cup \cdots \cup L(Q_k)$ for given queries $P, Q_1, \ldots, Q_k$ from CQ($\sim, \not\sim$), is reducible to containment for CQ($\sim, \not\sim$) queries.*

Proof The proof is analogous to the proof of Lemma 3 in [38] (and quite similar to the proof of Lemma 22). □

Theorem 25 *Containment for CQ($\sim, \not\sim$) queries is undecidable.*

Proof The proof is similar to the proof of Theorem 23 in that it modifies the proof of Theorem 21 and combines it with (the new) Lemma 24. Given an instance

$$(w_1, u_1), \ldots, (w_n, u_n)$$

of PCP over alphabet $\Gamma = \{a, b\}$, we construct a CQ($\sim, \not\sim$) query P and a disjunction Q of CQ($\sim, \not\sim$) queries, such that $P \subseteq Q$ if and only if $(w_1, u_1), \ldots,$ (w_n, u_n) has *no* solution.

As before, we let $\Sigma = \{r, \#\} \uplus I \uplus \Gamma$. 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 solution candidate, i.e., it should be a word 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 schema that determines the set of labels that can occur in trees, we cannot ensure that two nodes have the same label by a disjunction over all pairs of non-equal labels, as we did in the proof of Theorem 21. Therefore, we use data values to *encode* the labels of Σ.

We first describe how to achieve (1). Let $\Sigma = \{\sigma_1, \dots, \sigma_m\}$ with $\sigma_1 = r$ and $\sigma_m = \#$. The query P guarantees that the tree has a path that starts with positions with pairwise distinct data values which carries all symbols from $\Sigma - \{\#\}$, beginning with r, and has at least two occurrences of $\#$:

$$\begin{aligned} P \equiv \exists x_1, \dots, x_{m-1}, y_1, y_2 \ : \\ \bigwedge_{i=1}^{m-2} \mathit{Child}(x_i, x_{i+1}) \wedge \bigwedge_{i=1}^{m-1} \sigma_i(x_i) \wedge \bigwedge_{1 \le i < j \le m-1} x_i \not\sim x_j \\ \wedge \mathit{Child}^+(x_{m-1}, y_1) \wedge \mathit{Child}^+(y_1, y_2) \wedge \#(y_1) \wedge \#(y_2). \end{aligned}$$

As in the proof of Theorem 21, we use the query Q to ensure (ENC1)–(ENC6). Furthermore, to make sure that solution candidates do 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 $i \in \{1, \dots, n\}$ we must make sure that I_i is followed by w_i (if in the first half of the solution candidate) 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 word 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 $I_i \cdot w_i \cdot a$.

However, we can not do this directly, since, as already mentioned above, we can not exhaustively enumerate all labels that might occur in a tree. Therefore, we need to modify the queries described in the previous paragraph along the same lines as we modify the other queries from the proof of Theorem 21 to achieve (2), as explained next.

The idea for (2) is very simple. The new encoding uses two (consecutive) positions y, z to encode one position x of the old encoding of a solution candidate (with the exception of the $m-1$ first positions and the two $\#$-positions). Position y is responsible for the data value d of x and therefore just has d as data value. Position z encodes the label σ_j of position x by carrying the same data value as the j-th position of the path (i.e., the position to which x_j is mapped in P above). The labels of y and z are thus irrelevant. The queries Q_i and the queries that shall ensure the pattern of solution candidates have to be adapted accordingly, in a straightforward manner.

We note that the application of Lemma 24 to P and $Q_1, \dots, Q_k$ might introduce $\sim$-atoms in P'. □

6 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 (see, e.g., [6,44]) adhering to the grammar

$$\begin{array}{lll}
\text{locpath} & ::= & \text{'/' locpath} \mid \text{locpath '/' locpath} \mid \text{locstep} \\
 & & \text{locpath} \cap \text{locpath} \\
\text{locstep} & ::= & \text{axis '::' ntst '[' bexpr ']'} \ldots \text{'[' bexpr ']'} \\
\text{bexpr} & ::= & \text{bexpr 'and' bexpr} \mid \text{locpath.} \\
\text{axis} & ::= & \text{'self'} \mid \text{'child'} \mid \text{'parent'} \mid \text{'descendant'} \mid \\
 & & \text{'descendant-or-self'} \mid \text{'ancestor'} \mid \\
 & & \text{'ancestor-or-self'} \mid \text{'following'} \mid \\
 & & \text{'following-sibling'} \mid \text{'preceding'} \mid \\
 & & \text{'preceding-sibling'.}
\end{array}$$

where “locpath” is the start production and “ntst” denotes Σ-symbols labeling document nodes or the star ‘∗’ that matches all tags (“node tests”). All operators come from Core XPath 1.0, except for the *path intersection* operator ‘∩’ which is from XPath 2.0. The semantics of the path intersection operator can be found in [44]. 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 6 in Section 3.2.3, 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 9, using predicate expressions.

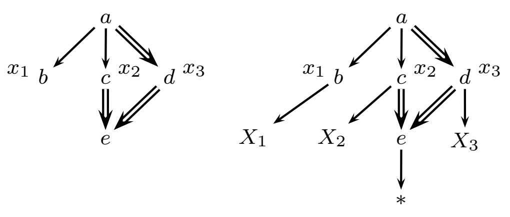


Fig. 12 How to reduce from n-ary queries to 0-ary queries.

7 Boolean versus N-ary Queries

Until now, we always considered conjunctive queries without free variables. This means that we only looked at whether a tree models the query or not instead of considering queries that return n-tuples. 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 Kimelfeld and Sagiv [30], 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 12. The reduction does two things. First of all, it introduces for each free variable x_i, a new variable x'_i and adds the atoms $\mathit{Child}(x_i, x'_i) \wedge X_i(x'_i)$ to the query, where X_i is a new label. Second, for each leaf node v of the query[8] that does not correspond to a free variable, it adds a new variable v' and adds the atom $\mathit{Child}(v, v')$ to the query. For example, for the query $P(x_1, x_2, x_3)$, we obtain the query P' on the right of Figure 12. Here, nodes u labeled with $*$ in the figure are nodes for which the query does not have a label, i.e., does not have an atom of the form $a(u)$. It is now easy to see that, for two queries $P(\overline{x})$ and $Q(\overline{x})$[9] 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 [30].

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

[8] For the purpose of the reduction, a node v of the query is a leaf node if and only if the query does not have any atom of the form $\mathit{Child}(v, w)$ or $\mathit{Child}^+(v, w)$.

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

8 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 [44], by David [20] (which evolved independently from ours), and by Lakshmanan et al. [32]. The connection with Hidders' work [29] is explained more elaborately in [7]. 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 are much more powerful. They 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 [44]). 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 27 in [44] when DTD information is considered. Ten Cate and Lutz consider Core XPath 2.0 queries with path intersection and vertical navigation (denoted CoreXPath$_{\downarrow,\uparrow}(\cap)$ in their paper) and prove that path containment w.r.t. DTD information is 2EXPTIME-complete (Theorem 27 and Proposition 6 in [44]). Their proof uses negation and union in queries, but our lower bound proof in Section 6 shows that, in the presence of DTDs, containment for CoreXPath$_{\downarrow,\uparrow}(\cap)$ queries is 2EXPTIME-hard, even when the queries do not use union or negation. (Without DTD information, containment of CoreXPath$_{\downarrow,\uparrow}(\cap)$ without union or negation drops to Π_2^P [7], since this fragment of CoreXPath$_{\downarrow,\uparrow}(\cap)$ is a subclass of conjunctive queries over trees.)

David studies the complexity of satisfiability for Boolean combinations of *data tree patterns* with respect to DTDs [20]. 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 [20] 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.

Conjunctive queries over trees are closely related to the tree patterns investigated in the context of incomplete XML [3, 21, 27]. The *incomplete trees* introduced in [3] are tree-shaped but have variables with which they can test data value equality. When investigated in a setting where one can express that each node carries a unique data value, a data value equality test between two nodes therefore expresses that the nodes are the same. For this reason, some of our proofs on queries with data value tests (Section 5) can be adapted to show similar results about incomplete trees (see, e.g., [21]). In particular, the proof of Theorem 8.1 in [21] builds on the proof of our Theorem 25.

In XML data exchange, the pattern queries for specifying the relationship between source data and target data are similar to conjunctive queries over trees [9,23,22]. The topic of XML data exchange is treated in depth in [2], Part 3.

Datalog programs that operate on trees and that natively use relations such as *Child* and *Descendant* are closely connected to conjunctive queries over trees as well. In fact, such Datalog programs are usually more powerful than conjunctive queries over trees and, therefore, our lower bound proofs can be used to obtain lower bounds for Datalog query containment, see, e.g., [4,1, 10].

Furthermore, there is a large amount of work on static analysis for XPath 1.0 (see, e.g., [5,18,19,26,37,38,40,47]). 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 indicated this in the paper whenever relevant.

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. [10] The results of the paper do not overlap much with our results on satisfiability, since they consider a limited, non-recursive, form of DTDs. However, their claim [32, Theorem 3.2] that query satisfiability for queries with structural constraints, Value Based Constraints (VBCs) and no wildcards is in PTIME, seems to be wrong in the light of our findings. Indeed, by adapting the proof of our Lemma 15, we can conclude that this problem is NP-hard, as shown next. We state the lemma in the terminology of [32] but the main construction in our NP-hardness proof for the same problem can be understood from our definitions.

Lemma 26 *Query satisfiability for queries with structural constraints, Value Based Constraints (VBCs) and no wildcards is NP-hard.*

Proof We give a reduction from Shortest Common Supersequence. Thereto, let S and k be an input of Shortest Common Supersequence. Let $S = \{b_1^1 \cdots b_{n_1}^1, \ldots, b_1^m \cdots b_{n_m}^m\}$ be a set of strings over some alphabet. Then the query Q is defined as shown in Figure 13. The idea is that a common supersequence for S must be formed in the data values of a length k string, that is enforced by the right hand side of Figure 13. The confluency in the bottom #-labeled node is obtained via *structural constraints*, which allow to identify

[10] Here, structural constraints include node identities and VBCs allow comparison of data values to constants.

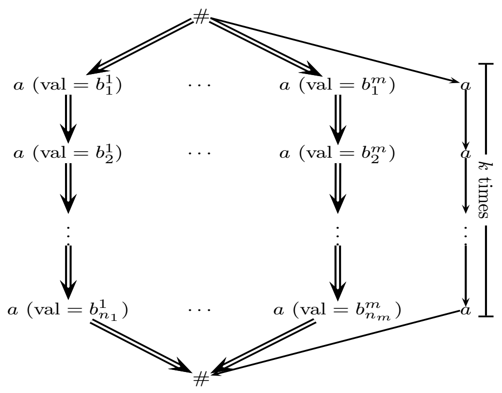


Fig. 13 Query Q for the proof of Lemma 26. Each single arrow denotes the *Child*-axis, and each double arrow denotes the *Child*$^+$-axis.

nodes (see [32]). 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. □

9 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 [44].

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 (see, e.g., [39]). 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 27 *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. □

Finally, we point out that some of our lower bound proofs (Theorems 6, 9, 12, and 15) use non-fixed alphabets. It is not yet clear if the proofs can also be adapted for alphabets of constant size.

Acknowledgments

This work was supported by grant number MA 4938/2–1 from the Deutsche Forschungsgemeinschaft (Emmy Noether Nachwuchsgruppe) and the Swedish Research Council grant 621-2011-6080.

References

1. S. Abiteboul, P. Bourhis, A. Muscholl, and Z. Wu. Recursive queries on trees and data trees. In *International Conference on Database Theory (ICDT)*, pages 93–104, 2013.
2. M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.
3. P. Barceló, L. Libkin, A. Poggi, and C. Sirangelo. XML with incomplete information. *J. ACM*, 58(1):4, 2010.
4. M. Benedikt, P. Bourhis, and P. Senellart. Monadic datalog containment. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 79–91, 2012.
5. M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55(2), 2007.
6. A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. XML Path Language (XPath) 2.0. Technical report, World Wide Web Consortium, January 2007. http://www.w3.org/TR/xpath20/.
7. H. Björklund, W. Martens, and T. Schwentick. Conjunctive query containment over trees. *J. Comput. Syst. Sci.*, 77(3):450–472, 2011.
8. H. Björklund, W. Martens, and T. Schwentick. Validity of tree pattern queries with respect to schema information. In *Mathematical Foundations of Computer Science (MFCS)*, pages 171–182, 2013.
9. M. Bojanczyk, L. A. Kolodziejczyk, and F. Murlak. Solutions in XML data exchange. *J. Comput. Syst. Sci.*, 79(6):785–815, 2013.
10. M. Bojanczyk, F. Murlak, and A. Witkowski. Containment of monadic datalog programs via bounded clique-width. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 427–439, 2015.
11. M. Bojanczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3), 2009.
12. A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Inf. Comput.*, 142(2):182–206, 1998.
13. A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
14. A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.
15. B. S. Chlebus. Domino-tiling games. *JCSS*, 32(3):374–392, 1986.
16. J. Clark and M. Murata. Relax NG specification. http://www.relaxng.org/spec-20011203.html, December 2001.

17. W. Czerwinski, C. David, K. Losemann, and W. Martens. Deciding definability by deterministic regular expressions. In *International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, pages 289–304. Springer, 2013.
18. W. Czerwinski, W. Martens, M. Niewerth, and P. Parys. Minimization of tree pattern queries. In *Symposium on Principles of Database Systems (PODS)*, pages 43–54, 2016.
19. W. Czerwinski, W. Martens, P. Parys, and M. Przybylko. The (almost) complete guide to tree pattern containment. In *Symposium on Principles of Database Systems (PODS)*, pages 117–130, 2015.
20. C. David. Complexity of data tree patterns over XML documents. In *MFCS*, pages 278–289, 2008.
21. C. David, A. Gheerbrant, L. Libkin, and W. Martens. Containment of pattern-based queries over data trees. In *International Conference on Database Theory (ICDT)*, pages 201–212, 2013.
22. C. David, P. Hofman, F. Murlak, and M. Pilipczuk. Synthesizing transformations from XML schema mappings. In *International Conference on Database Theory (ICDT)*, pages 61–71, 2014.
23. C. David, L. Libkin, and F. Murlak. Certain answers for XML queries. In *Symposium on Principles of Database Systems (PODS)*, pages 191–202, 2010.
24. Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.
25. J. Gallant, D. Maier, and J. A. Storer. On finding minimal length superstrings. *JCSS*, 20(1):50–58, 1980.
26. F. Geerts and W. Fan. Satisfiability of XPath queries with sibling axes. In *DBPL*, pages 122–137, 2005.
27. A. Gheerbrant, L. Libkin, and T. Tan. On the complexity of query answering over incomplete XML documents. In *ICDT*, pages 169–181, 2012.
28. G. Gottlob, C. Koch, and K. U. Schulz. Conjunctive queries over trees. *J. ACM*, 53(2):238–272, 2006.
29. J. Hidders. Satisfiability of XPath expressions. In *DBPL*, pages 21–36, 2003.
30. B. Kimelfeld and Y. Sagiv. Revisiting redundancy and minimization in an XPath fragment. In *Extending Database Technology (EDBT)*, pages 61–72, 2008.
31. P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. *JCSS*, 61(2):302–332, 2000.
32. L. V. S. Lakshmanan, G. Ramesh, H. Wang, and Z. Zhao. On testing satisfiability of tree pattern queries. In *VLDB*, pages 120–131, 2004.
33. P. Lu, J. Bremer, and H. Chen. Deciding determinism of regular languages. *Theory of Computing Systems*, pages 1–43, 2014.
34. W. Martens and F. Neven. On the complexity of typechecking top-down XML transformations. *Theoretical Computer Science*, 336(1):153–180, 2005.
35. W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM J. Comput.*, 39(4):1486–1530, 2009.
36. W. Martens, F. Neven, T. Schwentick, and G.J. Bex. Expressiveness and complexity of XML schema. *ACM Trans. Database Syst.*, 31(3):770–813, 2006.
37. M. Marx. Conditional XPath. *ACM TODS*, 30(4):929–959, 2005.
38. G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004.
39. F. Murlak, M. Oginski, and M. Przybylko. Between tree patterns and conjunctive queries: Is there tractability beyond acyclicity? In *Mathematical Foundations of Computer Science (MFCS)*, pages 705–717, 2012.
40. F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *LMCS*, 2(3), 2006.
41. E.L. Post. A variant of a recursively unsolvable problem. *Bull. AMS*, 52(4):264–268, 1946.
42. K.J. Räihä and E. Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *Theoretical Computer Science*, 16(2):187–198, 1981.
43. M. Takahashi. Generalizations of regular sets and their application to a study of context-free languages. *Inf. Control*, 27(1):1–36, 1975.
44. B. ten Cate and C. Lutz. The complexity of query containment in expressive fragments of XPath 2. *J. ACM*, 56(6), 2009.

45. James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

46. Moshe Y. Vardi. Reasoning about the past with two-way automata. In *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, pages 628–641, 1998.

47. P. T. Wood. Containment for XPath fragments under DTD constraints. In *ICDT*, 2003. Full version, obtained through personal communication.