Efficient Algorithms for the Tree Homeomorphism Problem

Michaela Götz[1], Christoph Koch[1], and Wim Martens[2]

[1] Saarland University
Saarbrücken, Germany
{goetz,koch}@infosys.uni-sb.de
[2] University of Dortmund
Dortmund, Germany
wim.martens@udo.edu

Abstract. Tree pattern matching is a fundamental problem that has a wide range of applications in Web data management, XML processing, and selective data dissemination. In this paper we develop efficient algorithms for the tree homeomorphism problem, i.e., the problem of matching a tree pattern with exclusively transitive (descendant) edges. We first prove that deciding whether there is a tree homeomorphism is LOGSPACE-complete, improving on the current LOGCFL upper bound. As our main result we develop a practical algorithm for the tree homeomorphism decision problem that is both space- and time efficient. The algorithm is in LOGDCFL and space consumption is strongly bounded, while the running time is linear in the size of the data tree. This algorithm immediately generalizes to the problem of matching the tree pattern against all subtrees of the data tree, preserving the mentioned efficiency properties.

1 Introduction

Tree patterns are a simple query language for tree-structured data. They are at the heart of several widely-used Web languages such as XPath and XQuery [4]. As a consequence, they form part of a number of typing mechanisms such as XML Schema, and of Web Programming Languages. They have also been used as query languages in their own right, for example for expressing subscriptions in publish-subscribe systems [1, 5, 6, 13].

The general tree pattern matching problem considered in the literature is the problem of finding a mapping between two node-labeled trees which is, in a sense, a cross of a subtree homomorphism and a homeomorphism. In this paper we consider a clean and important special case of the tree pattern embedding problem that we call the *tree homeomorphism problem*. The question we consider is whether there is a mapping θ from the nodes of the first tree, the *tree pattern* or *query*, to the nodes of the second tree, the *data tree*, such that if node y is a child of x in the first tree, then $\theta(y)$ is a *descendant* of $\theta(x)$ in the second tree. We also consider the *tree homeomorphism matching problem*: finding *all* nodes v of the data tree such that there is such a tree homeomorphism with v the image

	time	space	streaming
Yannakakis 1981 [19]	$O(\|Q\| \cdot \|D\| \cdot \mathit{depth}(D))$	$O(\mathit{depth}(Q) \cdot \|D\|)$	no
Gottlob et al. 2002 [10]	$O(\|Q\| \cdot \|D\|)$	$O(\|Q\| \cdot \|D\|)$	no
Olteanu et al. 2004 [15]	$O(\|Q\| \cdot \|D\| \cdot \mathit{depth}(D))$	$O(\|Q\| \cdot \mathit{depth}(D) + \|D\|)$	yes
Bar-Yossef et al. 2005 [3]	$O(\|Q\| \cdot \|D\|)$	$O(\|Q\| \cdot \log \|D\| + \mathit{cand}_D)$	yes
Ramanan 2005 [16]	$O((\|Q\| + \mathit{depth}(D)) \cdot \|D\|)$	$O(\|Q\| \cdot \mathit{depth}(D) + \mathit{cand}_D)$	yes
Our bottom-up algorithm	$O(\|Q\| \cdot \|D\| \cdot \mathit{depth}(\|Q\|))$	$O(\mathit{depth}(D) \cdot \mathit{branch}(D))$	no
Our LOGSPACE algorithm	$\mathit{poly}(\|Q\| + \|D\|)$	$O(\log(\|Q\| + \|D\|))$	no

Table 1. Time and space consumption for algorithms solving the tree homeomorphism matching problem. Here $\mathit{depth}(\cdot)$ and $\mathit{branch}(\cdot)$ denote the depth and maximal branching factor of a tree, respectively.

of the root node of the pattern tree. This problem of selecting all nodes whose subtrees match the tree pattern has frequent application in XML and Web query processing [1, 10].

While this problem is of immediate practical relevance and a substantial number of papers have studied complexity and efficient algorithms for tree pattern matching, the precise complexity of both the general tree pattern matching problem and the tree homeomorphism problem are open; they are both known to be in LOGCFL and LOGSPACE-hard [11].[3] The former can be immediately concluded from earlier results on the complexity of the acyclic conjunctive queries [12] and the positive navigational fragment of XPath [11], both much stronger languages. The latter is a direct consequence of the fact that reachability in trees is LOGSPACE-complete [8].

Much work has been dedicated to developing efficient algorithms for finding matches of tree patterns and tree homeomorphisms. Certain algorithms aim at processing the data tree as a stream (i.e., in a single scan) [5, 6, 13, 9, 15, 2, 3, 16]. For this case a number of lower bound results have been obtained using mechanisms from communication complexity [2, 3, 14]. It is basically known that streaming algorithms for even simple tree patterns consume space proportional to the size of the data tree in the worst case. Table 1 lists algorithms for the tree homeomorphism matching problem together with bounds on their running time and space consumption. Here D is the data tree and Q is the tree pattern. We assume a random access machine model with unit cost for reading and writing integers. Some of the algorithms presented support generalizations of the tree homeomorphism problem but where a better bound is known for the tree homeomorphism problem, it is shown. Some of the streaming algorithms [3, 16] use a notion of candidate node sets cand_D which depends on the algorithm and which can be of size close to $|D|$ in the worst case. The algorithm of [3] makes the assumption of so-called non-recursive data trees, in which no two nodes such that one is a descendant of the other may have the same label.

[3] A note to the reviewers that will be removed from the final version of the paper, but which is important here: The paper [17] claims the result that Core XPath with (only) the descendant axis is in LOGSPACE, but the proof is incorrect. The result is not present in the journal version of that work [11].

In this paper we study the tree homeomorphism (matching) problem. We establish a tight complexity characterization and develop an algorithm for the node-selection problem (shown at the bottom of Table 1) that is both time- and space efficient. In detail, the technical contributions of this paper are as follows.

- We first develop a top-down algorithm for the tree homeomorphism problem that is in LOGDCFL.
- From this we develop a proof that the problem is LOGSPACE-complete, improving on the LOGCFL upper bound from [11].
- As our main result we develop a bottom-up LOGDCFL algorithm for computing all solutions of the tree homeomorphism problem which is both time and space efficient. This is a rather difficult algorithm and the correctness proof is involved. The algorithm runs in time $O(|D| \cdot |Q| \cdot \mathit{depth}(Q))$ and employs a stack of depth bounded by $\mathcal{O}(\mathit{depth}(D) \cdot \mathit{branch}(D))$.
  The algorithm may be of relevance in practical implementations. Indeed, in most Web or XML applications, the data tree is *much* larger than the tree pattern yet its depth is rather small. It can be observed that ours is the only algorithm in Table 1 — and to the best of our knowledge, in existence — that can guarantee a space bound that does not contain the size, but only depth and branching factor, of the data tree as a term. At the same time the algorithm admits a good time bound.
  Furthermore, the algorithm is of relevance in theory as well. It is a first step in classifying the complexity of positive Core XPath with child and descendant axis, which is probably the most widely used XPath fragment in practice. Its precise complexity, however, is unknown.
- In some applications (e.g., for certain XML data trees), a few nodes can have a very large number of children. Our algorithm can be made to run in space $O(\mathit{depth}(D) \cdot \log(\mathit{branch}(D)))$ with the same time bound if we assume the data tree to be in a ranked form that can be obtained by a LOGSPACE linear-time preprocessing algorithm. Given that ours is an offline algorithm it means little loss of generality to assume that data trees are kept in a database in this preprocessed form.

The paper presents these result basically in the order given here. Because of space limitations, some proofs had to be moved to an appendix.

2 Definitions

By $\mathbb{N}$ we denote the set of strictly positive integers. By Σ we denote a finite alphabet. The set of *unranked Σ-trees*, denoted by $\mathcal{T}_\Sigma$, is the smallest set of strings over Σ and the parenthesis symbols "(" and ")" which contains the empty string and, for each $a \in \Sigma$ and $w \in (\mathcal{T}_\Sigma)^*$, contains $a(w)$. So, a tree is either ε (empty) or is of the form $a(T_1 \cdots T_n)$ where each T_i is a tree. In the tree $a(T_1 \cdots T_n)$, the subtrees $T_1, \ldots, T_n$ are attached to the root labeled a. When we write a tree as $a(T_1 \cdots T_n)$, we tacitly assume that every T_i is a non-empty tree. Moreover, we write a rather than $a()$. Notice that there is no a priori bound on

the number of children of a node in a Σ-tree; such trees are therefore *unranked*. A *hedge* H is a finite sequence $T_1 \cdots T_n$ of trees. Hence, the set of *unranked Σ-hedges*, denoted by $\mathcal{H}_\Sigma$, equals $(\mathcal{T}_\Sigma)^*$. When we write a hedge as $T_1 \cdots T_n$, we tacitly assume that every T_i is a non-empty tree. In the sequel, whenever we say tree or hedge, we always mean Σ-tree or Σ-hedge, respectively. We will slightly abuse terminology and use the term "tree" to also refer to a hedge consisting of one tree, and we use the term "hedge" to also refer to the union of trees and hedges. We assume familiarity with terms such as *child*, *parent*, *descendant*, *ancestor*, *leaf*, *root*, *first child*, *last child*, *first sibling*, and *last sibling*.

For a hedge H, the *set of nodes* or *domain* of H, denoted by $\mathrm{Dom}(H)$, is the subset of $\mathbb{N}^*$ inductively defined as follows: *(i)* if $H = \varepsilon$, then $\mathrm{Dom}(H) = \emptyset$; *(ii)* if $H = a$, then $\mathrm{Dom}(H) = \{1\}$; *(iii)* if $H = a(T_1 \cdots T_n)$, where each $T_i \in \mathcal{T}_\Sigma - \{\varepsilon\}$, then $\mathrm{Dom}(H) = \{1\} \cup \bigcup_{i=1}^{n} \{1iu \mid 1u \in \mathrm{Dom}(T_i)\}$; and *(iv)* if $H = T_1 \cdots T_n$ with $n \geq 2$ and each $T_i \in \mathcal{T}_\Sigma - \{\varepsilon\}$, then $\mathrm{Dom}(H) = \{iu \mid 1u \in \mathrm{Dom}(T_i)\}$. The label of node u in the tree or hedge H, denoted by $\mathrm{lab}^H(u)$, is defined as follows: *(i)* if $H = a$ and $u = 1$, then $\mathrm{lab}^H(u) = a$; *(ii)* if $H = a(T_1 \cdots T_n)$ and $u = 1iv$ with $i \in \{1, \ldots, n\}$, then $\mathrm{lab}^H(u) = \mathrm{lab}^{T_i}(1v)$; and *(iii)* if $H = T_1 \cdots T_n$ with $n \geq 2$ and $u = iv$ with $i \in \{1, \ldots, n\}$, then $\mathrm{lab}^H(u) = \mathrm{lab}^{T_i}(1v)$.

By $|H|$, we denote the number of nodes in a hedge H. The *depth* of a node u in hedge H, denoted by $\mathrm{depth}^H(u)$, is 1 when $u \in \mathbb{N}$ and $1 + \mathrm{depth}(v)$ when $u = vi$ and $i \in \mathbb{N}$. The *height* of a node u in hedge H, denoted by $\mathrm{height}^H(u)$, is 1 when u is a leaf and $\max(\mathrm{height}^H(u1), \ldots, \mathrm{height}^H(uk)) + 1$ when u has $k > 0$ children. By $\mathrm{subtree}^H(u)$, we denote the subtree of H rooted at node u. In the remainder of the paper, we usually leave H implicit when H is clear from the context.

The Tree Homeomorphism Problem. A *tree pattern query* (with descendant edges) Q is an unranked tree over the alphabet $\Sigma \uplus \{*\}$. In the following, we use the terms *data tree* or *data hedge* to refer to ordinary Σ-trees and Σ-hedges. Given a data hedge H, a node $u \in \mathrm{Dom}(H)$, and a tree pattern query Q, we say that H *matches* Q *at node* u, denoted by $H \models^u Q$, if one of the following holds:

- $H = a$, $Q = a$ or $Q = *$, and $u = 1$;
- $H = a(T_1 \cdots T_n)$, $Q = a$ or $Q = *$, and $u = 1$;
- $H = a(T_1 \cdots T_n)$, $T_i \models^{1v} Q$, and $u = 1iv$, for some $i \in \{1, \ldots, n\}$;
- $H = T_1 \cdots T_n$, $T_i \models^{1v} Q$, and $u = iv$, for some $i \in \{1, \ldots, n\}$;
- $H = a(T_1 \cdots T_n)$, $Q = x(Q_1 \cdots Q_m)$, $u = 1$, $x \in \Sigma \uplus \{*\}$, $a \models^1 x$, and, for every $k = 1, \ldots, m$, there exists an $i_k \in \{1, \ldots, n\}, u_k \in \mathrm{Dom}(T_{i_k})$, such that $T_{i_k} \models^{u_k} Q_k$.

Notice that the ordering of children in our tree pattern queries does not matter. This corresponds to the well known semantics of XPath queries with descendant axes [7]. In the following, we abbreviate by $H \models Q$ that $H \models^u Q$ for some $u \in \mathrm{Dom}(H)$. Alternatively, we say that H *matches* Q.

In this paper, we are interested in the following problems. Given a data tree T and a tree pattern query Q, the *tree homeomorphism problem* consists of deciding whether $T \models Q$. Furthermore, we are interested in *computing all answers* for the

Algorithm 1 Tree pattern matching with descendant axes: Top-down algorithm MATCH

```
    MATCH (DNode d, QNode q)
 2: if d matches q then
      return ∀ child q_c of q ∃ child d_c of d: MATCH(d_c,q_c)
 4: else                                   ▷ q not matched yet, try d's children
      return ∃ child d_c of d: MATCH(d_c,q)
 6: end if
```

tree homeomorphism problem, that is, computing all nodes $u \in \mathrm{Dom}(T)$ such that $T \models^u Q$. We refer to the latter problem as *tree homeomorphism matching*.

We assume that trees are stored on tape as a set of records; one for each node. Each record contains a pointer to its first child, last child, parent, previous sibling, and next sibling.

In the remainder of the paper, we assume a fixed data tree D and a fixed query tree Q for ease of presentation. We will refer to nodes of D and Q as *data nodes* and *query nodes*, respectively.

3 A Top-Down Algorithm

This section provides a simple top-down algorithm for the tree homeomorphism matching problem. The core of this top-down algorithm lies in a simple procedure that decides, given a data node d and a query node q, whether subtree(d) $\models$ subtree(q).

3.1 A Top-Down LOGDCFL Algorithm

Algorithm 1 describes the procedure MATCH to test whether subtree(d) $\models$ subtree(q). It is straightforward to prove that MATCH is indeed correct.

Lemma 1. MATCH *is correct. That is, given a data node d and a query node q,* MATCH *returns true iff subtree(d) $\models$ subtree(q).*

A proof of this lemma is provided in the Appendix.

We can turn the procedure in Algorithm 1 into an algorithm TOP-DOWN-MATCH for the tree homeomorphism matching problem as follows. First, we need a procedure EXACT-MATCH that, given a data node d and query node q, decides whether subtree(d) $\models^1$ subtree(q). This is easy: EXACT-MATCH only differs from MATCH in l.5, where it just returns false. Given a data node d and the root q_{root} of the query tree, TOP-DOWN-MATCH now simply iterates over all the data nodes and returns every data node d for which EXACT-MATCH(d, q_{root}) returns true. From this construction and from the correctness of MATCH, it is now immediate that TOP-DOWN-MATCH is correct as well.

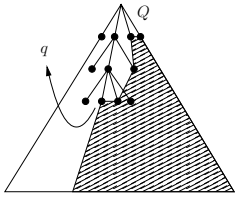


Fig. 1. Illustration of the remainder of q in Q.

Time and Space Complexity. It can be shown quite directly that the time complexities of Match and Exact-Match are in $\mathcal{O}(|\text{subtree}(d)| \cdot |\text{subtree}(q)|)$. As Top-Down-Match simply calls Exact-Match for every data node, we immediately have the following result, which is proved in the Appendix.

Proposition 2. *The running time of* Top-Down-Match *is in* $\mathcal{O}(|D|^2 \cdot |Q|)$. *Moreover,* Top-Down-Match *makes* $\mathcal{O}(|D|^2 \cdot |Q|)$ *comparisons between a data node and a query node.*

It is immediate from our implementation of the algorithm that it can be executed by a deterministic logarithmic space bounded auxiliary pushdown automaton (see, e.g., [18]). Moreover, by Proposition 2, this auxiliary pushdown automaton runs in polynomial time. It follows from [18] that the tree homeomorphism matching problem is in LOGDCFL. As the maximum recursion depth of Algorithm 1 is $\mathcal{O}(\text{depth}(D))$, this renders the algorithm quite space-efficient, but the running time being quadratic in the size of the data tree, and the many unnecessary comparisons between query and data nodes are quite unsatisfactory. In the next section, we show how these issues can be resolved by turning to a bottom-up approach.

3.2 A LOGSPACE Procedure

While the top-down algorithm does not seem to be well-suited for efficiently computing *all* nodes u for which $D \models^u Q$, it is quite useful for *deciding* whether $D \models Q$, from a complexity theory point of view. Indeed, as we will exhibit, a modified version of Match can decide in LOGSPACE whether $D \models Q$.

To this end, we assume the *left-to-right pre-order* ordering on nodes in trees and hedges in the remainder of this section. In particular, for every node u with k children in a hedge H, we have that $u < u1 < u2 < \cdots < uk$. For a node u, we denote by $u + 1$ the next node in the depth first, left-to-right traversal.

We argue how to transform Algorithm 1 into a LOGSPACE algorithm that decides whether $D \models Q$. Intuitively, the LOGSPACE algorithm processes the data and query trees in a top-down left-to-right manner, just like Algorithm 1, but the essential difference lies in a backtracking procedure. When, for example, Algorithm 1 matches a query node q onto some data node d and discovers that d's subtree does *not* match q's subtree, it uses the recursion stack to find the

Algorithm 2 LOGSPACE decision procedure: Top-down algorithm L-MATCH. We assume left-to-right preordering on trees.

```
    L-MATCH (DNode d, QNode q)
 2: if d matches q, and both d and q have children then
      return L-MATCH (d+1,q+1)
 4: else if d does not match q and d has a child then
      return L-MATCH (d+1, q)
 6: else if d matches q and q is a leaf then
      if q is maximal in Q then return true   ▷ none of q's ancestors has a right sib.
 8:   else
        d' ← BACKTRACK(d, q+1)     ▷ node onto which q+1's parent was matched
10:     return L-MATCH (d'+1, q+1)
      end if
12: else                          ▷ d is a leaf and (d does not match q or q is not a leaf)
      if d is maximal in D then return false
14:   end if
      while q has a parent do
16:     d' ← BACKTRACK(d, q)              ▷ node onto which q's parent was matched
        if d' is an ancestor of d+1 then return L-MATCH (d+1, q)
18:     else q ← q.parent
        end if
20:   end while
      return L-MATCH (d+1, q)
22: end if
```

data node d' onto which q's parent was matched in the data tree. Instead of using this recursion stack, the LOGSPACE algorithm enters a subprocedure BACKTRACK(d, q) that *recomputes* d'. Essentially, BACKTRACK(d, q) computes the highest possible node d'' on the path from D's root to d, such that the path from D's root to d'' matches the path from Q's root to q's parent. The crux of the algorithm is that this is *correct*, i.e., $d'' = d'$; and that BACKTRACK(d, q) can be performed using only logarithmic space on a Turing Machine.[4]

We present the LOGSPACE algorithm in Algorithm 2. For ease of presentation, we have written the algorithm as a recursive procedure, but it can be implemented to only use logarithmic space. This can be seen by observing Algorithm 2: every recursive call to L-MATCH is a return-statement, so the algorithm does not change when the recursion stack is not used at all.

Let, for a query node q, the *remainder of q in Q* be the subhedge of Q consisting of the nodes $\{q' \mid q \leq q' \leq q_{\max}\}$, where $q_{\max}$ is the maximal query nodes w.r.t. the depth-first left-to-right ordering. We illustrate the remainder of q in Q in Figure 1. Given a data node d and a query node q, the algorithm first

[4] It should be noted that this routine is very time-inefficient on a logspace Turing Machine. As we cannot store the paths from d (resp., q) to the root of D (resp., Q), it involves a quadratic number of depth-first left-to-right runs through the data (resp., query tree) to test whether d (resp., q) is still a descendant of a given data node (resp., query node).

tries to match the remainder of q in Q consistently with what has already been matched in D (lines 2–11). If this fails, it either returns false (line 13), or enters a backtracking procedure (lines 15–21).

Lemma 3. *Algorithm 2 is correct. That is, given the roots d and q of a data D and query tree Q, Algorithm 2 decides whether $D \models Q$. Moreover, Algorithm 2 only uses logarithmic space.*

Theorem 4. *The tree homeomorphism problem is LOGSPACE-complete.*

4 The Bottom-up Algorithm

Although the previously presented top-down algorithms for tree homeomorphism matching are quite space-efficient, their time complexity is quite high and they involve quite a lot of recomputing of already obtained matchings, which is unsatisfactory. We therefore turn to a bottom-up matching approach which has the property that *no* obtained matchings between the data and query tree need to be recomputed, which leads to a better time complexity of the overall algorithm.

Before presenting the bottom-up algorithm for the tree homeomorphism matching problem in detail, we need to introduce several formal notions. As in the previous section, we first present an algorithm for the tree homeomorphism problem and then show how to change it into an algorithm the for the tree homeomorphism matching problem.

In the present section, we assume the *left-to-right post-order* ordering on nodes in trees and hedges. In particular, for every node u with k children in a hedge H, we have that $u1 < u2 < \cdots < uk < u$. For a node u, we denote by $u+1$ the next node in the left-to-right postorder traversal. Hence, when we, e.g., use terminology such as "largest" and "smallest", we always assume the left-to-right post ordering. In this section, we also assume that XML documents are stored on tape in left-to-right postorder (or, alternatively, together with a left-to-right postorder index), which allows a random-access machine model to verify the left-to-right post-order ordering in constant time. For technical purposes, we also assume two dummy nodes in every tree and hedge: nil and ∞. The node nil is such that nil$+1$ is the smallest node in the hedge, and the node ∞ is defined as the successor of the largest node of the hedge. Given two nodes $h_{\text{from}} \leq h_{\text{until}}$ in a hedge H, we denote by the interval $[h_{\text{from}}, h_{\text{until}}]$ the subhedge of H consisting only of the nodes $\{v \mid h_{\text{from}} \leq v \leq h_{\text{until}}\}$. The notion of such an interval in a tree is illustrated in Figure 2(a). Here, the interval $[h_{\text{from}}, h_{\text{until}}]$ is the striped area in the tree. Given a hedge H and a node $h \in \text{Dom}(H)$, we denote by $\text{subhedge}^H(h)$ the subhedge $[h_{\text{from}}, h]$, where h_{from} is the smallest descendant of h's leftmost sibling according to the left-to-right postorder ordering. We illustrate this notion in Figure 2(b).

When H is a data hedge or a tree pattern query, we refer to $[h_{\text{from}}, h_{\text{until}}]$ as a data or query hedge interval, respectively. We extend the semantics of tree pattern matching to hedges as follows. Let $Q_1 \cdots Q_n$ be a query hedge interval $[q_{\text{from}}, q_{\text{until}}]$ and $D_1 \cdots D_m$ be a data hedge interval $[d_{\text{from}}, d_{\text{until}}]$. We say that

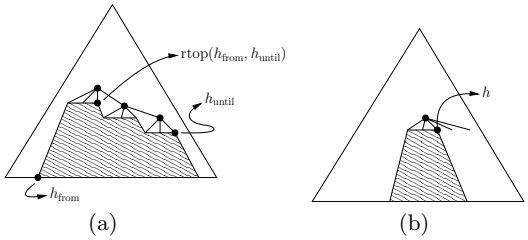


Fig. 2. Illustration of a hedge interval and RTop (left) and of subhedge$^H(h)$ (right).

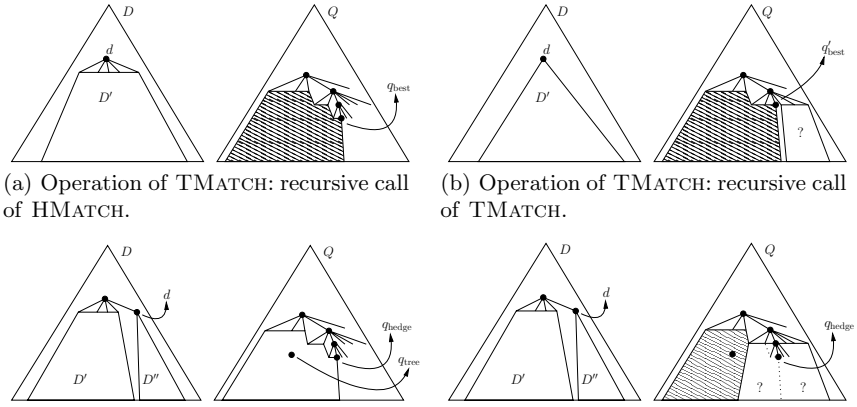
$[d_{\text{from}}, d_{\text{until}}]$ matches $[q_{\text{from}}, q_{\text{until}}]$, denoted by $[d_{\text{from}}, d_{\text{until}}] \models [q_{\text{from}}, q_{\text{until}}]$, if, for every Q_i, $i = 1, \ldots, n$, there exists a D_j, $j = 1, \ldots, m$, such that $D_j \models Q_i$.

Before presenting the intuition about the bottom-up tree homeomorphism algorithm, we describe an auxiliary procedure RTop, which, given two nodes h_{from} and h_{until}, returns the rightmost node among the topmost nodes in the interval $[h_{\text{from}}, h_{\text{until}}]$. More formally, RTop$(h_{\text{from}}, h_{\text{until}})$ is the node u such that depth(u) is minimal and u is larger than every other node v in $[h_{\text{from}}, h_{\text{until}}]$ with depth(u) = depth(v). This notion is illustrated in Figure 2(a). Furthermore, in order to simplify the presentation of the algorithm, we define RTop$(h_{\text{from}}, h_{\text{until}}) = \infty$ if $h_{\text{from}} > h_{\text{until}}$. Notice that RTop can easily be computed in time linear in the depth of the tree and in logarithmic space by traversing the path from h_{until} to the query root and comparing the previous siblings of nodes on the path with h_{from} w.r.t. the left-to-right post-ordering. Indeed, assume that $h_{\text{from}} \leq h_{\text{until}}$. Let u be the highest ancestor of h_{until} that has a previous sibling s such that $s \geq h_{\text{from}}$. If no such u exists, then rtop$(h_{\text{from}}, h_{\text{until}})$ is h_{until}. Otherwise, rtop$(h_{\text{from}}, h_{\text{until}})$ is s.

We first present an algorithm for *deciding* whether $D \models Q$ and show later how it can be extended to an algorithm for the tree homeomorphism matching problem. The main procedure of our algorithm is called TMatch. Given a data node d and query nodes q_{from} and q_{until}, TMatch returns the largest query node q in the interval $[q_{\text{from}}, q_{\text{until}}]$ such that subtree$^D(d)$ matches $[q_{\text{from}}, q]$ if q exists; and $q_{\text{from}} - 1$ otherwise. Hence, if d is the root of D, and q_{from} and q_{until} are the leftmost leaf and the root of Q, respectively, then $D \models Q$ if and only if TMatch returns q_{until}.

TMatch uses an auxiliary procedure called HMatch, which, given a data node d and query nodes q_{from} and q_{until}, returns the largest node q in the interval $[q_{\text{from}}, q_{\text{until}}]$ such that subhedge$^D(d)$ matches $[q_{\text{from}}, q]$ if q exists; and $q_{\text{from}} - 1$ otherwise.

We start by explaining the operation of TMatch, which is presented in Algorithm 3. Given a data node d and query nodes q_{from} and q_{until}, TMatch first starts by recursively calling HMatch with the same query nodes for the subhedge D' of D defined by d's last child, yielding result q_{best} (see Figure 3(a)). In the remainder of TMatch, we essentially want to test how q_{best} can be improved when we also consider the node d in addition to D'. One particular



(a) Operation of TMatch: recursive call of HMatch.

(b) Operation of TMatch: recursive call of TMatch.

(c) Operation of HMatch: first recursive calls of TMatch and HMatch.

(d) Operation of HMatch: a subsequent recursive call of TMatch, trying to improve q_{tree}.

Fig. 3. Illustrations of the tree homeomorphism algorithm.

interesting case is when q_{best} is a last sibling and its parent has the same label as d. In this case, we can at least improve our best query node to q_{best}'s parent which we call here q'_{best}. Furthermore, it is possible that q'_{best} is not yet the best query node we can obtain. In particular, we still need to test which part of the hedge defined by $[q'_{\text{best}}+1, q'_{\text{best}}.\text{lastSibling}]$ can be matched in the subtree below d (see Figure 3(b)). The largest node that is obtained in this manner is the node that TMatch should return.

We now explain the operation of HMatch, which is presented in Algorithm 4. Essentially, given d, q_{from}, and q_{until}, HMatch starts by recursively calling itself with the same query nodes on the hedge defined by the previous sibling of d (i.e., D' in Figure 3(c)), yielding q_{hedge}, and by calling TMatch with the same query nodes on the subtree under d itself (D'' in Figure 3(c)), yielding q_{tree}. The remainder of HMatch consists of iteratively improving q_{tree} and q_{hedge}. That is, while it is possible that D' and D'' yield small values of q_{tree} and q_{hedge}, their concatenation can give rise to a much larger part of the query that can be matched. Essentially, this is due to the fact that the matching of tree pattern queries is *unordered*. For example, it can occur that we need to match a certain first sibling in D', a second one in D'', a third one again in D' and so on. Hence, the procedure HMatch alternates between finding best matches in D' and D'' until it reaches a fixpoint.

However, we need to take care in how this fixpoint is computed. One possible case is illustrated in Figure 3(d). This particular case builds further on the situation in Figure 3(c). Here, we try to improve q_{tree} by starting the TMatch procedure again for the node d, but now only with the part of the query marked with question marks. The case where q_{tree} is larger than q_{hedge} is dual and not illustrated here.

Algorithm 3 Tree pattern matching: function TMatch.

```
    TMatch (DNode d, QNode q_from, QNode q_until)
 2: if d is a leaf then q_best ← q_from − 1
    else q_best ← HMatch(d.lastChild, q_from, q_until)
 4: end if
    if q_best + 1 ≤ q_until and d matches q_best + 1 then
 6:   q_best ← q_best + 1
      if q_best + 1 ≤ q_best.lastSib then
 8:     return TMatch(d, q_best + 1, q_best.lastSib)
      else return q_best
10:   end if
    else return q_best
12: end if
```

Example 5. Figure 4(a) and 4(b) illustrate an example for the bottom up algorithm. For brevity, we denote TMatch and HMatch with TM and HM, respectively. The first calls of TM and HM demonstrate the basic recursive structure of our algorithm: TM on a node d calls HM on the rightmost child of d. HM on a node d returns TM of d if that node is a first sibling; or performs a divide-and-conquer technique by calling HM on the left sibling of d and TM on d itself (as in the function call $\mathrm{HM}(d_4, q_1, q_5)$). Further recursive calls to TM or HM are then needed to maximize the part of the query that can be matched.

The simplest function call in the example that performs such further recursive calls is the call $\mathrm{HM}(d_2, q_1, q_5)$, which starts by computing $q_{\mathrm{hedge}} = \mathrm{HM}(d_1, q_1, q_5)$ and $q_{\mathrm{tree}} = \mathrm{TM}(d_2, q_1, q_5)$. As can be seen in Figure 4(b), $q_{\mathrm{hedge}} = \mathrm{nil}$. The call $\mathrm{TM}(d_2, q_1, q_5)$ is more successful, because d_2 and q_1 are both labeled with a. In general, it might be possible that q_2 and further nodes can be matched in subtree(d_2). The function call $\mathrm{TM}(d_2, q_2, q_4)$ checks that possibility. (For sure, q_1 and q_5 cannot both be matched on d_2, which is why we restrict the query tree interval by q_4.) But q_2 is not labeled with a so the return value of the two TM calls is q_1. After this initial phase, $\mathrm{HM}(d_2, q_1, q_5)$ tries to improve q_{tree} and q_{hedge} iteratively. It calls $\mathrm{HM}(d_1, q_2, q_4)$ and improves q_{hedge} to be q_2, because q_2 and d_1 are both labeled with b. Further improvements fail as there is no c-labeled node in the subhedge of d_2.

A similar iterative improvement is illustrated by $\mathrm{HM}(d_3, q_1, q_5)$. Observe that we try to improve q_{tree} here and call $\mathrm{TM}(d_4, q_2, q_4)$ and $\mathrm{TM}(d_4, q_3, q_3)$. Only the latter call yields an improvement. But we cannot omit the former one: if subtree(d_4) would match subtree(q_4), then the former call would yield q_4 and the latter call would yield q_3. As we want our algorithm to return the largest query node such that the interval ending with it can be matched the result of the former call would have been the relevant one in that case.

Correctness. The main technical difficulty of the paper is proving that TMatch is correct. A full proof of the following Lemma can be found in the Appendix.

Algorithm 4 Tree pattern matching: function HMatch.

```
    HMatch (DNode d, QNode q_from, QNode q_until)
14: if d is a first sibling then return TMatch(d, q_from, q_until)
    else
16:   q_hedge ← HMatch(d.prevSib, q_from, q_until)
      q_tree ← TMatch(d, q_from, q_until)
18:   loop
        if q_hedge = q_tree then return q_hedge
20:     else if q_tree < q_hedge then
          rtop ← RTop(q_tree + 1, q_hedge)
22:       while rtop < ∞ and q_hedge < rtop.lastSib do
            q_tree ← TMatch(d, rtop+1, rtop.lastSib)
24:         rtop ← RTop(q_tree + 1, q_hedge)
          end while
26:       if q_tree ≤ q_hedge then return q_hedge
          end if
28:     else
          rtop ← RTop(q_hedge + 1, q_tree)
30:       while rtop < ∞ and q_tree < rtop.lastSib do
            q_hedge ← HMatch(d.prevSib, rtop + 1, rtop.lastSib)
32:         rtop ← RTop(q_hedge + 1, q_tree)
          end while
34:       if q_hedge ≤ q_tree then return q_tree
          end if
36:     end if
      end loop
38: end if
```

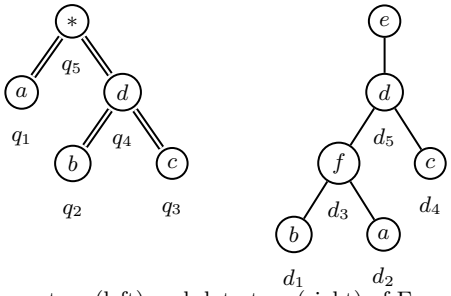
Lemma 6. *Let D be a data tree and let Q be a query tree.* TMatch *is correct, that is, given the root node d of D, the smallest and largest node q_{from} and q_{until} of Q, respectively,* TMatch *returns q_{until} iff $D \models Q$.*

We now argue how TMatch can be modified to a procedure TMatch-All, that computes *all* data nodes u such that $D \models^u Q$. In order to compute *all* the matches, we add a test to l.9 of TMatch. That is, before returning q_{best}, we test whether q_{best} is the root of Q, and we output d if it is. Now we return $q_{\text{best}} - 1$, as if the query root was not matched. Furthermore, TMatch-All recursively calls TMatch-All and HMatch-All instead of TMatch and HMatch. Here HMatch-All is the same as HMatch, except that it recursively calls TMatch-All and HMatch-All instead of HMatch and TMatch.

The following theorem can be proved:

Theorem 7. *Let d be the root node of D and let q_{from} be the smallest and q_{root} be the largest node of Q, respectively.* TMatch-All *is correct, that is,* TMatch-All(d, q_{from}, q_{until}) *outputs the data nodes u such that $D \models^u Q$.*

Proof (Sketch). It follows directly from our additional test and the correctness of TMatch that $D \models^u Q$ for all the nodes u that TMatch-All outputs.



(a) Query tree (left) and data tree (right) of Example 5.

$\text{TM}(d_6, q_1, q_5) \Rightarrow q_5$
$\text{HM}(d_5, q_1, q_5) \Rightarrow q_4$
$\text{TM}(d_5, q_1, q_5) \Rightarrow q_4$
$\text{HM}(d_4, q_1, q_5) \Rightarrow q_3$
$\text{HM}(d_3, q_1, q_5) \Rightarrow q_2$
$\text{TM}(d_3, q_1, q_5) \Rightarrow q_2$
$\text{HM}(d_2, q_1, q_5) \Rightarrow q_2$
$\text{HM}(d_1, q_1, q_5) \Rightarrow \text{nil}$
$TM(d_1, q_1, q_5) \Rightarrow \text{nil}$
$\text{TM}(d_2, q_1, q_5) \Rightarrow q_1$
$TM(d_2, q_2, q_4) \Rightarrow q_1$
$\text{HM}(d_1, q_2, q_4) \Rightarrow q_2$
$TM(d_1, q_2, q_4) \Rightarrow q_2$
$TM(d_1, q_3, q_3) \Rightarrow q_2$
$\text{TM}(d_2, q_3, q_3) \Rightarrow q_2$
$\text{TM}(d_4, q_1, q_5) \Rightarrow \text{nil}$
$\text{TM}(d_4, q_2, q_4) \Rightarrow q_1$
$\text{TM}(d_4, q_3, q_3) \Rightarrow q_3$

(b) Function calls of HMatch (HM) and TMatch (TM) of Example 5.

Fig. 4. Illustrations for Example 5.

It remains to prove that, if $D \models^u Q$, then TMatch-All outputs u. Towards a contradiction, assume that there is an u such that $D \models^u Q$, but u was not reported by TMatch-All. By an easy induction it can be shown that for every data node d_0 in D there is a call TMatch-All for d_0's subtree and Q. In partic-

ular, there was a call TMatch-All$(u, q_{\text{from}}, q_{\text{root}})$. Since this call did not output u, it follows that u must have children and that HMatch-All$(u.\text{lastChild}, q_{\text{from}}, q_{\text{root}}) < q_{\text{root}} - 1$, (because otherwise q_{root} and u would have been compared and u would have been written to the output). In general, we have that HMatch-All$(d, q_1, q_2) = \min(\text{HMatch}(d, q_1, q_2), q_{\text{root}} - 1)$.
It follows that HMatch-All$(u.\text{lastChild}, q_{\text{from}}, q_{\text{root}}) =$ HMatch$(u.\text{lastChild}, q_{\text{from}}, q_{\text{root}})$.

If we now call TMatch$(u, q_{\text{from}}, q_{\text{root}})$, it calls HMatch$(u.\text{lastChild}, q_{\text{from}}, q_{\text{root}})$, which yields again a value less than $q_{\text{root}} - 1$. Therefore, the return value of TMatch$(u, q_{\text{from}}, q_{\text{root}})$ is less than q_{root}. But we assumed that subtree$(u) \models Q$, which contradicts the correctness of TMatch proved in Lemma 6. □

Time and Space Complexity. First, we need to show that our algorithm determines in PTIME whether $D \models Q$. Notice that the naïve manner of computing the running time of TMatch gives rise to only an exponential upper bound. Indeed, define *(i)* $T(N)$ as the running time of TMatch on d, q_{from}, and q_{until}, where subtree(d) and $[q_{\text{from}}, q_{\text{until}}]$ have N nodes in total, and *(ii)* $H(N)$ as the running time of HMatch on d, q_{from}, and q_{until}, where subhedge(d) and $[q_{\text{from}}, q_{\text{until}}]$ have N nodes in total. Then, we have that $T(2) \leq p(N)$ for a polynomial p, $T(N) \leq p(N) + H(N-1) + T(N-1)$, and $H(N) \leq T(N) + X(N)$, where $X(N) \geq 0$. Hence, $T(N) \leq 2^{N-1}$, which is obviously not sufficient.

We therefore employ a slightly more sophisticated approach in the following Lemma. Its proof can be found in the Appendix.

Lemma 8. *Given the root node of a data tree D, and the smallest and largest query nodes and of a query tree Q, respectively,* TMatch *runs in time $\mathcal{O}(|D| \cdot |Q| \cdot \text{depth}(Q))$. Moreover,* TMatch *makes $\mathcal{O}(|D| \cdot |Q|)$ comparisons between a data node and a query node.*

The depth(Q) factor in the complexity of TMatch is due to the calls to rtop in HMatch, and the computation of the successors of query nodes.

Theorem 9. TMatch-All(D, Q) *runs in time $\mathcal{O}(|D| \cdot |Q| \cdot \text{depth}(Q))$. Moreover,* TMatch-All *makes $\mathcal{O}(|D| \cdot |Q|)$ comparisons between a data node and a query node.*

Currently, the maximum recursion depth of TMatch-All is $\mathcal{O}(\text{depth}(D) \times \text{branch}(D))$, where branch$(D)$ is the maximum number of children a node in D has. We have the branch(D) factor because HMatch$(d, q_{\text{from}}, q_{\text{until}})$ calls HMatch$(d.\text{prevSib}, q_{\text{from}}, q_{\text{until}})$. However, this bound can be improved using a simple preprocessing step: we can turn D into a binary tree D_{bin} by inserting intermediate levels of special nodes between each data node and its children. By doing so, D only grows linearly in size and the depth only grows by a factor of log(branch(D)).

As Q only uses descendant axes, we have that $D \models^u Q$ iff $D_{\text{bin}} \models^u Q$.[5] When this preprocessing step is carried out, our algorithm still has $\mathcal{O}(|D||Q|\text{depth}(Q))$

[5] Under the assumption that the new dummy nodes do not match $*$, which can be trivially incorporated in the algorithm.

time complexity, but the recursion/stack depth is improved to $\mathcal{O}(\text{depth}(D) \cdot \log(\text{branch}(D)))$.

References

1. M. Altinel and M. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proc. VLDB*, pages 53–64, 2000.
2. Z. Bar-Yossef, M. Fontoura, and V. Josifovski. On the memory requirements of XPath evaluation over XML streams. In *Proc. PODS*, pages 177–188, 2004.
3. Z. Bar-Yossef, M. Fontoura, and V. Josifovski. Buffering in query evaluation over XML streams. In *Proc. PODS*, 2005.
4. N. Bruno, D. Srivastava, and N. Koudas. Holistic twig joins: Optimal XML pattern matching. In *Proc. SIGMOD*, pages 310–321, 2002.
5. C. Y. Chan, W. Fan, P. Felber, M. N. Garofalakis, and R. Rastogi. Tree pattern aggregation for scalable XML data dissemination. In *Proc. VLDB*, pages 826–837, 2002.
6. C. Y. Chan, P. Felber, M. N. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. In *Proc. ICDE*, pages 235–244, 2000.
7. J. Clark and S. DeRose. XML Path Language (XPath). Technical report, World Wide Web Consortium, November 1999. http://www.w3.org/TR/xpath.
8. S. A. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *J. Algorithms*, 8:385–394, 1987.
9. Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. Fischer. Path sharing and predicate evaluation for high-performance XML filtering. *ACM Trans. Database Syst.*, 28(4):467–516, 2003.
10. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005.
11. G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. The complexity of XPath query evaluation and XML typing. *J. ACM*, 52(2):284–335, 2005.
12. G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(1):431–498, 2001.
13. T. J. Green, A. Gupta, G. Miklau, M. Onizuka, and D. Suciu. Processing XML streams with deterministic automata and stream indexes. *ACM Trans. Database Syst.*, 29(4):752–788, 2004.
14. M. Grohe, C. Koch, and N. Schweikardt. Tight lower bounds for query processing on streaming and external memory data. In *Proc. ICALP 2005*, 2005.
15. D. Olteanu, T. Furche, and F. Bry. An evaluation of regular path expressions with qualifiers against XML streams. In *Proc. BNCOD 2004*, pages 31–44, 2004.
16. P. Ramanan. Evaluating an XPath query on a streaming XML document. In *Proc. COMAD 2005*, pages 41–52, 2005.
17. L. Segoufin. Typing and querying XML documents: some complexity bounds. In *Proc. PODS*, pages 167–178. ACM Press, 2003.
18. I. H. Sudborough. Time and tape bounded auxiliary pushdown automata. In *Proc. MFCS 1977*, pages 493–503. Springer Verlag, 1977.
19. M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. VLDB*, pages 82–94, 1981.

A Appendix

This appendix is provided for the convenience of the reviewers and does not form part of the paper. It provides some proofs for which there was no space in the body of the paper.

A.1 Proofs for Section 3.1

Proof of Lemma 1. MATCH *is correct. That is, given a data node* d *and a query node* q, MATCH *returns true iff* $\mathit{subtree}(d) \models \mathit{subtree}(q)$.

Proof. By induction over the size of the data tree, denoted by n.
$\underline{n = 1:}$ We have that $\text{subtree}(d) = a$ for some a. We return true, if and only if the query tree consists of one node and d matches this node. The correctness follows from the tree pattern matching definition, which says that if $\text{subtree}(d) = a$, $\text{subtree}(q) = a$ or $\text{subtree}(q) = *$, $\text{subtree}(d) \models \text{subtree}(q)$.
$\underline{n > 1:}$ We consider two cases.

- If d matches q, we return true if, for every child q_c of q, there exists a child d_c of d such that $\text{MATCH}(d_c, q_c)$ returns true. If the query tree consists of only one node, this is obviously correct. If q has children, the correctness follows from the induction hypothesis and the last item in the definition of tree pattern matchings: If $\text{subtree}(d) = a(T_1 \cdots T_n)$, $\text{subtree}(q) = x(Q_1 \cdots Q_m)$, $x \in \Sigma \uplus \{*\}$, $a \models x$, and, for every $k = 1, \ldots, m$, there exists an $i_k \in \{1, \ldots, n\}$, such that $T_{i_k} \models Q_k$, then $\text{subtree}(d) \models \text{subtree}(q)$. If there exists a q_c such that $\text{MATCH}(d_c, q_c)$ is false for every d_c, we would also fail to match the whole query tree into a subtree of a child of d. Again by the definition of tree pattern matchings it is then correct to return false.
- If d does not match q, we test whether there is a child d_c of d such that $\text{subtree}(q)$ can be matched into $\text{subtree}(d_c)$. By the induction hypothesis, the recursive calls of $\text{MATCH}(d_c, q)$ compute this correctly. If there is such a matching, it is correct to return true by the definition of tree pattern matchings: If $\text{subtree}(d) = a(T_1 \cdots T_n)$ and $T_i \models \text{subtree}(q)$, then $\text{subtree}(d) \models \text{subtree}(q)$. Furthermore, if $\text{subtree}(d) = a(T_1 \cdots T_n)$, d does not match q, and there does not exist a T_i such that $T_i \models \text{subtree}(q)$, then, by definition, $\text{subtree}(d) \not\models \text{subtree}(q)$. Hence, it is correct to return false. □

Time and Space Complexity.

Proof of Proposition 2. *The running time of* TOP-DOWN-MATCH *is in* $\mathcal{O}(|D|^2|Q|)$. *Moreover,* TOP-DOWN-MATCH *makes* $\mathcal{O}(|D|^2|Q|)$ *comparisons between a data node and a query node.*

We start by proving some simple observations about MATCH.

Observation 10.

1. $\text{MATCH}(d, q)$ *compares each node in* $\mathit{subtree}(d)$ *at most once with each node in* $\mathit{subtree}(q)$.

2. *The running time of* $\textsc{Match}(d, q)$ *is* $|\mathit{subtree}(d)| \cdot |\mathit{subtree}(q)|$.

Proof. 1. This is immediate from the fact that $\textsc{Match}(d, q)$ does not call $\textsc{Match}(d', q')$ twice, for any descendant d' of d and any descendant q' of q; and the fact that d' and q' are only compared to one another at the start of the call $\textsc{Match}(d', q')$.

2. This is an easy induction on $|\text{subtree}(d)|$. If $|\text{subtree}(d)| = 1$, then we test whether d matches q and discover that there are no children of d to iterate over. Hence, the running time is in $\mathcal{O}(|\text{subtree}(q)|)$.
   If $|\text{subtree}(d)| > 1$, then we test whether d matches q and we either execute $\textsc{Match}$ recursively for every child d_c of d and every child q_c of q; or we execute $\textsc{Match}$ recursively for every child d_c of d and q. In both cases, we can apply the induction hypothesis. In the first case, the time complexity becomes $\mathcal{O}(\sum_{q_c}(\sum_{d_c}(|\text{subtree}(d_c)| \cdot |\text{subtree}(q_c)|)))$, and in the second case, the time complexity becomes $\mathcal{O}(\sum_{d_c}(|\text{subtree}(d_c)| \cdot |\text{subtree}(q)|))$. Hence, both cases are in $\mathcal{O}(|\text{subtree}(d)| \cdot |\text{subtree}(q)|)$. □

Proposition 2 is now immediate from Observation 10 and the fact that Top-Down-Match calls $\textsc{Exact-Match}(d, q_{\text{root}})$ for every node d of D and the root node q_{root} of Q.

Algorithm 5 LOGSPACE decision procedure: Top-down algorithm L-MATCH.

```
    L-MATCH (DNode d, QNode q)
 2: if d matches q, and both d and q have children then                ▷ θ(q) = d
      return L-MATCH (d + 1,q + 1)
 4: else if d does not match q and d has a child then
      return L-MATCH (d + 1, q)
 6: else if d matches q and q is a leaf then                           ▷ θ(q) = d
      if q is maximal then
 8:     return true
      else
10:     d′ ← BACKTRACK(d, q + 1)      ▷ node onto which q + 1.parent was matched
        return L-MATCH (d′ + 1, q + 1)
12:   end if
    else                          ▷ d is a leaf and (d does not match q or q is not a leaf)
14:   if d is maximal then
        return false
16:   end if
      q′ ← q
18:   while q′ has a parent do
        d′ ← BACKTRACK(d, q′)              ▷ node onto which q′.parent was matched
20:     if d′ is an ancestor of d + 1 then
          return L-MATCH (d + 1, q′)
22:     else q′ ← q′.parent
        end if
24:   end while
      return L-MATCH (d + 1, q′)
26: end if
```

A.2 Proofs for Section 3.2

Recall that we assume the *left-to-right pre-order* ordering on nodes in trees. For a node u, we denote by $u + 1$ the next node in the depth first, left-to-right traversal.

Correctness of L-Match.

We want to show that L-MATCH returns true on input D and Q if and only if $D \models Q$. To simplify the analysis, we imaginarily extend the algorithm by defining a matching θ: Whenever the algorithm is executed we pull out paper and pen to write down θ. If the algorithm compares the labels of d and q in the function call L-MATCH(d, q) and they agree (in line 2, 6), we set $\theta(q) = d$ (and may overwrite older assignments). We present this in Algorithm 5.

This mapping θ is merely used to simplify the reasoning about the algorithm, we can make use of our mapping θ and prove properties about it. These properties can refer to snapshots during the execution of the actual algorithm.

Soundness. In this section we want to prove that whenever L-MATCH returns true on input D and Q, then $D \models Q$. In fact we prove a stronger claim: If L-MATCH returns true, then our mapping θ is a label preserving matching that obeys the descendant requirement. Hence, $D \models Q$.

In order to prove the soundness of L-MATCH, we first show the following Lemma:

Lemma 11. *Let D be a data tree and Q be a query tree. Further, let* L-MATCH(d, q) *be a function call resulting from the initial procedure call* L-MATCH(*root*(D), *root*(Q)). *Then at the time when* L-MATCH(d, q) *is called*

(1) the restriction of θ to query nodes lesser than q is a label preserving matching that obeys the descendant requirement,
(2) θ matches the path $\langle q.\text{parent} \cdots \text{root}(Q)\rangle$ into the path $\langle d.\text{parent} \cdots \text{root}(D)\rangle$ as high as possible, and
(3) the path $\langle q \cdots \text{root}(Q)\rangle$ cannot be matched into the path $\langle d.\text{parent} \cdots \text{root}(D)\rangle$.

Proof. Proof by induction on the position k of L-MATCH(d, q) in the series of function calls resulting from the initial procedure call L-MATCH(root(D), root(Q)).

$k = 1$: For L-MATCH(root(D), root(Q)) there is nothing to show.

$k > 1$: Let the claim be true for the first k function calls. Let L-MATCH(d, q) be the k^{th} function call. We prove that it is also true for the $k + 1^{\text{th}}$ function call (if there is one). We consider four cases according to Algorithm 2.

- If the labels of d and q agree and both nodes have children (line 2), the next function call is L-MATCH($d + 1$, $q + 1$), where $d + 1$ and $q + 1$ are the leftmost children of d and q, respectively. We know by induction hypothesis that θ, restricted to query nodes less than q, is a label preserving matching that obeys the descendant requirement. We extend this mapping by $\theta(q) = d$. This mapping is clearly label-preserving. We need to show that $\theta(q)$ is a descendant of $\theta(q.\text{parent})$. But this clear, since by induction hypothesis $\langle q.\text{parent} \cdots \text{root}(Q)\rangle$ is matched as high as possible into the path $\langle d.\text{parent} \cdots \text{root}(D)\rangle$, which proves (1). Combining this with the fact that $\langle q \cdots \text{root}(Q)\rangle$ cannot be matched into $\langle d.\text{parent} \cdots \text{root}(D)\rangle$ we conclude that $\langle q \cdots \text{root}(Q)\rangle$ is matched as high as possible into $\langle d \cdots \text{root}(D)\rangle$, which proves (2). With the descendant requirement it then follows that the path $\langle d \cdots \text{root}(D)\rangle$ cannot match the path $\langle q + 1 \cdots \text{root}(Q)\rangle$, which proves (3).
- If the labels of d and q do not agree and d has children (line 4), the next function call is L-MATCH($d + 1$, q), where $d + 1$ is the leftmost child of d. We do not extend θ in that case and all requirements (1), (2), and (3) follow from the induction hypothesis.
- If the labels of d and q agree and q is a leaf and q is not maximal (line 6), we extend the mapping θ by $\theta(q) = d$. As in the first case of this proof we know by induction hypothesis that θ, restricted to query

nodes less than q, is a label preserving matching that obeys the descendant requirement. The extended θ is still label preserving and still obeys the descendant requirement, because, due to the induction hypothesis, $\langle q.\text{parent} \cdots \text{root}(Q)\rangle$ is matched into the path $\langle d.\text{parent} \cdots \text{root}(D)\rangle$. Hence, (1) is true.

BACKTRACK$(d, q{+}1)$ calculates the highest ancestor d' of the data node d such that $\langle d' \cdots \text{root}(D)\rangle$ matches $\langle q+1.\text{parent} \cdots \text{root}(Q)\rangle$. Why does d' exist? First, note that $q{+}1.\text{parent}$ is an ancestor of q due to the left-to-right pre-order ordering. Second, by induction hypothesis $\langle q.\text{parent} \cdots \text{root}(Q)\rangle$ can be matched into the path $\langle d.\text{parent} \cdots \text{root}(D)\rangle$. Putting both facts together, the sub-path $\langle q+1.\text{parent} \cdots \text{root}(Q)\rangle$ can still be matched into the path $\langle d.\text{parent} \cdots \text{root}(D)\rangle$. Hence, d' exists and $d'+1$ is its leftmost child.

The next function call is L-MATCH$(d'{+}1, q{+}1)$. By induction hypothesis, the mapping θ matches the path $\langle q.\text{parent} \cdots \text{root}(Q)\rangle$ into the path $\langle d.\text{parent} \cdots \text{root}(D)\rangle$ as high as possible and therefore, θ also matches the sub-path $\langle q+1.\text{parent} \cdots \text{root}(Q)\rangle$ as *high* as possible into that data path. Due to BACKTRACK, d' is an ancestor of d, such that $\langle d' \cdots \text{root}(D)\rangle$, matches $\langle q+1.\text{parent} \cdots \text{root}(Q)\rangle$. Putting both facts together, we have (2): θ matches the sub-path $\langle q{+}1.\text{parent} \cdots \text{root}(Q)\rangle$ as high as possible into the path $\langle d' \cdots \text{root}(D)\rangle$.

The fact that d' is the *highest* ancestor of d such that $\langle d' \cdots \text{root}(D)\rangle$ matches $\langle q+1.\text{parent} \cdots \text{root}(Q)\rangle$ implies that $\langle d' \cdots \text{root}(D)\rangle$ does not match $\langle q+1 \cdots \text{root}(Q)\rangle$. Hence, (3) is proven.

– If d is a leaf and d is not maximal and (the labels of d and q do not agree or q has children) (line 13, we know that we have to try to match q somewhere else. We do not extend θ (but we might restrict θ), so we still have a label-preserving mapping that obeys the descendant requirement and (1) is true. We consider two cases.
  - First, assume that the next function call is L-MATCH$(d+1,\ q')$ in line 25. Then q' has no parent, $q' = \text{root}(Q)$, and (2) is trivially true. To prove (3), e.g. to prove that $\text{root}(Q)$ cannot be matched into the path $\langle d+1.\text{parent} \cdots \text{root}(D)\rangle$, we consider two cases.
    * If $q = q' = \text{root}(Q)$, by induction hypothesis $\text{root}(Q)$ cannot be matched into $\langle d.\text{parent} \cdots \text{root}(D)\rangle$ and therefore also not into the sub-path $\langle d+1.\text{parent} \cdots \text{root}(D)\rangle$.
    * If $q \neq q' = \text{root}(Q)$, then $\text{root}(Q)$ is an ancestor of q. By induction hypothesis we have that the mapping θ matches $\text{root}(Q)$ onto some ancestor of d and we also have that $\text{root}(Q)$ cannot be matched into the path $\langle\theta(\text{root}(Q)).\text{parent} \cdots \text{root}(D)\rangle$. Furthermore, the fact that we did not return a function call in line 21 implies that then $\theta(\text{root}(Q))$ is a descendant of $d+1.\text{parent}$. Putting both facts together, we conclude that $\text{root}(Q)$ cannot be matched into the path $\langle d+1.\text{parent} \cdots \text{root}(D)\rangle$
  - Now, assume that the next function call is L-MATCH$(d+1,\ q')$ in line 21. BACKTRACK$(d,\ q')$ has calculated the highest ancestor d'

of the data node d such that $\langle d' \cdots \text{root}(D)\rangle$ matches $\langle q'.\text{parent} \cdots \text{root}(Q)\rangle$. Why does d' exist? First, note that q' has a parent (line 20) and that q' is an ancestor or self of q. Hence, $q'.\text{parent}$ is an ancestor of $q.\text{parent}$. Further note, that by induction hypothesis the path $\langle d.\text{parent} \cdots \text{root}(D)\rangle$ matches the path $\langle q.\text{parent} \cdots \text{root}(Q)\rangle$ and therefore it also matches the sub-path $\langle q'.\text{parent} \cdots \text{root}(Q)\rangle$. It follows that d' exists and that $d' + 1$ is its leftmost child.
We know that q' is the lowest ancestor or self of q such that d' is an ancestor of $d + 1$ (observe the while loop). In fact, q' and its parent exist, because otherwise we would not make the next function call in line 21, but in line 25. Next, we will prove (2). By induction hypothesis the mapping θ matches the query path $\langle q.\text{parent} \cdots \text{root}(Q)\rangle$ and therefore also the sub-path $\langle q'.\text{parent} \cdots \text{root}(Q)\rangle$ as high as possible into the data path $\langle d.\text{parent} \cdots \text{root}(D)\rangle$. It follows that the mapping θ matches the path $\langle q'.\text{parent} \cdots \text{root}(Q)\rangle$ as high as possible into the sub-path $\langle d' \cdots \text{root}(D)\rangle$ due to Backtrack. With the fact that d' is an ancestor of $d + 1$ (line 21) it then follows that the mapping θ matches the path $\langle q'.\text{parent} \cdots \text{root}(Q)\rangle$ as high as possible into the path $\langle d + 1.\text{parent} \cdots \text{root}(D)\rangle$, which proves (2).
In order to prove (3), e.g. to prove that the path $\langle d + 1.\text{parent} \cdots \text{root}(D)\rangle$ cannot match the path $\langle q' \cdots \text{root}(Q)\rangle$, we consider two cases:

* If $q = q'$, by induction hypothesis the path $\langle d.\text{parent} \cdots \text{root}(D)\rangle$ cannot match the path $\langle q \cdots \text{root}(Q)\rangle$. Due to the left-to-right pre-order the path $\langle d + 1.\text{parent} \cdots \text{root}(D)\rangle$ is a sub-path of $\langle d.\text{parent} \cdots \text{root}(D)\rangle$. The claim follows.
* If $q \neq q'$, recall that q' is the lowest ancestor of q such that $\theta(q'.\text{parent})$ is an ancestor of $d + 1$ (observe the while loop and recall that by induction hypothesis $d' = \theta(q'.\text{parent})$). It follows, that q' is matched somewhere on the path from $d.\text{parent}$ to (but not including) $d + 1.\text{parent}$. By Lemma 11 we cannot match the path $\langle q' \cdots \text{root}(Q)\rangle$ higher. Hence, the path $\langle d + 1.\text{parent} \cdots \text{root}(D)\rangle$ does not match the path $\langle q' \cdots \text{root}(Q)\rangle$.

– Otherwise there does not follow a function call. □

Proposition 12. *Algorithm 2 is sound. That is, given a data D and query tree Q, if Algorithm 2 returns true, then $D \models Q$.*

Proof. If L-Match(d, q) returns true in line 8, we know that q is maximal (line 7) and that the labels of q and d agree (line 6). By Lemma 11 θ is a label-preserving mapping, which matches every node in $Q \backslash \{q\}$ with respect to the descendant requirement into D, such that q's parent is matched onto some ancestor of d. We extend the mapping by $\theta(q) = d$, and conclude that $D \models Q$.

Completeness. In this section we want to prove that whenever L-MATCH returns false on input D and Q, then $D \not\models Q$. In order to prove the completeness, we first show the following Lemma:

Lemma 13. *Let D be a data tree and let Q be a query tree. Let* L-MATCH(d, q) *be a function call resulting from the initial procedure call* L-MATCH(*root*(D), *root*(Q)). *Then, it holds for all left siblings $\hat{d}$ on the path from d to (but not including) θ(q.parent) or, in case q has no parent, the path from d to root(D) that*

$$\mathit{subtree}(\hat{d}) \not\models \mathit{subtree}(q).$$

Proof. Note, that by Lemma 11 we can talk about the image under θ of query nodes less than q. The proof is by induction on the position k of L-MATCH(d, q) in the series of function calls resulting from the initial procedure call L-MATCH(root(D), root(Q)).

$k = 1$**:** For L-MATCH(root(D), root(Q)) there is nothing to show, because there are no left siblings on the path $\langle \text{root}(D) \rangle$.

$k > 1$**:** Let the claim be true for the first k function calls. Let L-MATCH(d, q) be the k^{th} function call. We prove that it is also true for the $k+1^{\text{th}}$ function call (if there is one). We consider four cases according to Algorithm 5.

- If the labels of d and q agree and both nodes have children (line 2), $\theta(q)$ is said to be d. The next function call is L-MATCH($d+1$, $q+1$), where $d+1$ and $q+1$ are the leftmost children of d and q, respectively.
  The path from $d+1$ to (but not including) $\theta(q+1.\text{parent})$ is the path from $d+1$ to (but not including) its parent d. Now, $d+1$ has no left sibling, so there is nothing to show.
- If the labels of d and q do not agree and d has children (line 4), the next function call is L-MATCH($d+1$, q), where $d+1$ is the leftmost child of d.
  By induction hypothesis we know that no there is no left sibling $\hat{d}$ on the path from d to (but not including) $\theta(q.\text{parent})$ or the path from d to root(D) in case q has no parent, such that subtree($\hat{d}$) matches subtree(q). If we extend the path by d's leftmost child, the left siblings on the larger path are still the same. So, there is nothing left to show.
- If the labels of d and q agree and q is a leaf (line 6) and q is not maximal, $\theta(q)$ is said to be d.
  BACKTRACK($d, q+1$) calculates the highest ancestor d' of the data node d such that $\langle d' \cdots \text{root}(D) \rangle$ matches $\langle q+1.\text{parent} \cdots \text{root}(Q) \rangle$. By Lemma 11 we have that $\theta(q+1.\text{parent}) = d'$.
  The next function call is L-MATCH($d'+1$, $q+1$). The path from $d'+1$ to (but not including) $\theta(q+1.\text{parent})$ is the path from $d'+1$ to (but not including) its parent d'. Now, $d'+1$ has no left sibling, so there is nothing to show.
- If d is a leaf and (the labels of d and q do not agree or q has children) (line 13) and d is not maximal, then subtree(d) does not match subtree(q). We first show the following invariant we will need later:

Invariant 14. *Whenever the body of the while loop in line 18 is executed without returning a function call in line 21, it follows for the current q' that the subtree($\theta(q'.parent)$) does not match the subtree($q'.parent$).*

Proof. We prove the claim by induction over the number of executions of the while body, denoted by l.

$l = 1$**:** Here $q' = q$, q has a parent (line 18), and we know that:

- the subtree(q) cannot be matched into the subtree(d) (line 13),
- q cannot be matched into the path from d.parent to (but not including) $\theta(q.\text{parent})$ by Lemma 11,
- there are no right siblings on the path from d to (but not including) $\theta(q.\text{parent})$ (otherwise we would have returned a function call in line 21),
- the subtree(q) cannot be matched into the subtree($\hat{d}$) for every left sibling $\hat{d}$ of the path from d to (but not including) $\theta(q.\text{parent})$ by the main induction hypothesis.

Hence, no subtree of $\theta(q.\text{parent})$ matches subtree(q), which implies that the subtree($\theta(q.\text{parent})$) does not match the subtree($q.\text{parent}$).

$l > 1$**:** Let the claim be true for the first l while loop executions. We prove, that it is also true for the $l + 1^{\text{th}}$ execution.

Let q' be the query node of the $l{+}1^{\text{th}}$ while loop execution. Here, $q' \neq q$ and q' has a parent (line 18). There must have been a function call L-Match($q', \theta(q')$) and there must have been a while loop execution with the child of q' on the path from q to q' as current node. We know that:

- the subtree(q') cannot be matched into the subtree($\theta(q')$), by the induction hypothesis,
- q' cannot be matched into the path from $\theta(q')$.parent to (but not including) $\theta(q'.\text{parent})$ by Lemma 11,
- there are no right siblings on the path from $\theta(q')$ to (but not including) $\theta(q'.\text{parent})$ (otherwise we would have returned a function call in line 21),
- the subtree(q') cannot be matched into the subtree($\hat{d}$) for every left sibling $\hat{d}$ of the path from $\theta(q')$ to (but not including) $\theta(q'.\text{parent})$ by the main induction hypothesis.

Hence, no subtree of $\theta(q'.\text{parent})$ matches subtree(q'), which implies that the subtree($\theta(q'.\text{parent})$) does not match the subtree($q'.\text{parent}$). □

Now, we come back to the proof of the main induction. We consider two cases.

- First, assume that next function call is L-Match($d{+}1$, q') in line 25. Here q' is the query root. We need to show that there is no left sibling $\hat{d}$ on the path $\langle d{+}1 \cdots \text{root}(D)\rangle$, such that subtree($\hat{d}$) $\models$ subtree(q'). We consider two cases:
  - If $q = q' = \text{root}(Q)$, by induction hypothesis the subtree(q) cannot be matched into the subtree($\hat{d}$) of some left sibling $\hat{d}$ of

the path $\langle d \cdots \text{root}(D)\rangle$. Since $d+1$.parent is an ancestor of d, it is enough to show that the subtree(previous sibling$(d+1)$), which is the left sibling of $d+1$ that includes d, does not match the subtree(q). We know that

- the subtree(q) cannot be matched into the subtree(d),
- q cannot be matched into the path $\langle d.\text{parent} \cdots \text{root}(D)\rangle$ by Lemma 11,
- there are no right siblings on the path from d to (but not including) the previous sibling$(d+1)$ due to the left-to-right pre-order,
- the subtree(q) cannot be matched into the subtree($\hat{d}$) for every left sibling $\hat{d}$ of the path from d to root(D) by induction hypothesis.

It follows, that we cannot match the subtree(q) into the subtree(previous sibling$(d+1)$) at all.

∗ If $q \neq q' = \text{root}(Q)$, then by Lemma 11 there must have been a function call L-Match$(q', \theta(q'))$. By induction hypothesis the subtree(q') cannot be matched into the subtree($\hat{d}$) of some left sibling $\hat{d}$ of the path $\langle \theta(q') \cdots \text{root}(D)\rangle$.
Furthermore, there must have been a while loop execution with q''s child on the path from q to q' as current query node. Since $d+1$.parent is an ancestor of $\theta(q')$ (otherwise we would have returned a function call in line 21), it is enough to show that the subtree(previous sibling$(d+1)$), which is the left sibling of $d+1$ that includes d, does not match the subtree(q'). We know that:

- the subtree(q') cannot be matched into the subtree($\theta(q')$) by the invariant,
- q' cannot be matched into the path from $\theta(q')$.parent to root(D) by Lemma 11,
- there are no right siblings on the path from $\theta(q')$ to (but not including) the previous sibling$(d+1)$, because $d+1$.parent is an ancestor of $\theta(q')$, which is an ancestor of d,
- the subtree(q') cannot be matched into the subtree($\hat{d}$) for every left sibling $\hat{d}$ of the path from $\theta(q')$ to root(D) by the induction hypothesis.

It follows, that we cannot match the subtree(q') into the subtree(previous sibling$(d+1)$) at all.

- Now, assume that the next function call is L-Match$(d+1, q')$ in line 21. Backtrack(d, q') has calculated the highest ancestor d' of the data node d such that $\langle d' \cdots \text{root}(D)\rangle$ matches $\langle q'.\text{parent} \cdots \text{root}(Q)\rangle$. By Lemma 11 d' equals $\theta(q'.\text{parent})$.
We know that q' is the lowest ancestor or self of q such that $\theta(q'.\text{parent})$ is an ancestor of $d+1$ (observe the while loop). It follows, that q' is matched somewhere on the path from d to (but not including) $d+1$.parent (for the case $q' \neq q$). No matter whether $q' = q$ or not, there was a function call L-Match$(q', \bar{d})$ for some $\bar{d}$ on the path

from d to (but not including) $d+1$.parent. By induction hypothesis and Lemma 11 there is no left sibling $\hat{d}$ on the path from $\bar{d}$ to (but not including) $\theta(q'.\text{parent})$ such that subtree($\hat{d}$) matches subtree(q'). Since $\bar{d}$ is in the subtree(previous sibling($d+1$)), we now only need to show that subtree(previous sibling($d+1$)) does not match the subtree(q').

We consider two cases:

* If $q = q'$, then we know that:
  - the subtree(q) cannot be matched into the subtree(d),
  - q cannot be matched into the path from d.parent to (but not including) $\theta(q.\text{parent})$ by Lemma 11,
  - there are no right siblings on the path from d to (but not including) the previous sibling($d+1$) due to the left-to-right pre-order,
  - the subtree(q) cannot be matched into the subtree($\hat{d}$) for every left sibling $\hat{d}$ of the path from d to (but not including) $\theta(q.\text{parent})$ by induction hypothesis .

  It follows, that subtree(previous sibling($d+1$)) does not match the subtree(q').
* If $q \neq q'$, there must have been a while loop execution with q''s child on the path from q to q' as current query node and there must have been a function call L-Match($\theta(q')$, q'). We know that:
  - the subtree(q') cannot be matched into the subtree($\theta(q')$) by the invariant,
  - q' cannot be matched into the path from $\theta(q')$.parent to (but not including) $\theta(q.\text{parent})$ by Lemma 11,
  - there are no right siblings on the path from $\theta(q')$ to (but not including) the previous sibling($d+1$), because $d+1$.parent is an ancestor of $\theta(q')$, which is an ancestor of d,
  - the subtree(q') cannot be matched into the subtree($\hat{d}$) for every previous sibling $\hat{d}$ of the path from $\theta(q')$ to (but not including) $\theta(q.\text{parent})$ by the induction hypothesis.

  It follows, that we cannot match the subtree(q') into the subtree(previous sibling($d+1$)) at all.

– Otherwise there does not follow a function call. □

Proposition 15. *Algorithm 2 is complete. That is, given a data D and query tree Q, if Algorithm 2 returns false, then $D \not\models Q$.*

Proof. We consider two cases.

– Let $|D| = 1$. L-Match(root(D), root(Q)) returns true, if root(Q) is a leaf with an appropriate label in line 8 and false otherwise in line 16, which proves the completeness for that case.
– Now, let $|D| > 1$. Assume L-Match returns false in line 16. Let d and q be the nodes, such that in the execution of L-Match(d, q) false was returned.

Due to line 13, d is a leaf and (q has children or the labels of q and d do not agree). Due to line 14, d is the maximal node, which means that there are no right siblings on the path from d to the root.
Consider a slight modification of the data tree: We attach to the root of the data tree a child on the right. Its value in the left-to-right pre-order is now $d+1$, the highest value of nodes in the data tree. Call this tree D'. Observe from the algorithm, that replacing D by D' does not make any difference in the function calls before L-MATCH(d, q), because the algorithm traverses the data tree due to the left-to-right pre-order. However, in the function call L-MATCH(d, q) the algorithm would not return false anymore, instead it would call L-MATCH($d+1$, q') for some query node q'. By Lemma 15 we know that for every child d' of the data root in D, the subtree(d') cannot match the subtree(q'). We consider two cases.

- Assume that q' has a parent. It is clear that if there was a matching from Q into D, we would be able to match the subtree(q') into some subtree of the data root. But we are not able to do this, so $D \not\models Q$.
- Assume that q' is the query root. By Lemma 11 we know that we cannot match the query root into the path $\langle d{+}1.\text{parent} \cdots \text{root}(D)\rangle$. Hence, the labels of the query root and the data root do not agree and if there was a matching from Q into D, we would be able to match the subtree(q') into some subtree of the data root. But we are not able to do this, so $D \not\models Q$. □

Propositions 11 and 15 imply the correctness of L-MATCH.

Proposition 16. *Algorithm 2 is correct. That is, given the roots d and q of a data D and query tree Q,* L-MATCH(d, q) *decides whether $D \models Q$.*

Space Complexity of L-Match. We already argued in the main body of the paper that the recursion stack has no influence on the operation of L-MATCH. It remains to argue why BACKTRACK only needs logarithmic space. BACKTRACK(d, q) calculates the highest ancestor d' of the data node d such that the path $\langle d' \cdots \text{root}(D)\rangle$ matches the path $\langle q.\text{parent} \cdots \text{root}(Q)\rangle$. The difficulty lies in the fact that we cannot store both paths. Instead, we store d and q. So, we start at the root nodes and compare their labels. If they match, we determine their children that lie on the paths. (This is performed by iterating over the children c and deciding whether d, respectively q, is still in the subtree below c, which can be performed in LOGSPACE by a depth-first left-to-right traversal.) Otherwise, we determine only the child of the data node. We continue until we matched the whole path $\langle q.\text{parent} \cdots \text{root}(Q)\rangle$. Finally we return the data node, onto which we matched q.parent.

The Complexity of the Tree Homeomorphism Problem. As argued above, L-MATCH can be performed in LOGSPACE. Putting this together with the fact that reachability in trees is LOGSPACE-complete, given the tree as a pointer structure, we obtain Theorem 4.

Theorem 4. *The tree homeomorphism problem is LOGSPACE-complete.*

A.3 Proofs for Section 4

Correctness Proof Our first purpose is to prove Lemma 6:
Proof of Lemma 6. *Let D be a data tree and let Q be a query tree.* TMatch *is correct, that is, given the root node d of D, the smallest and largest node q_{from} and q_{until} of Q, respectively,* TMatch *returns q_{until} if and only if $D \models Q$.*

We start with a few simple observations.

Observation 17. *A node u is not a last sibling $\Leftrightarrow$ $u+1$ is a leaf.*

Proof. Left to right: if u is not a last sibling, then $u+1$ is the leftmost descendant leaf of the right sibling of u, or the right sibling of u itself if it is a leaf. Right to left: if u is the last node in a left-to-right postorder traversal, then u is a last sibling for which $u+1$ does not exist. For all other last siblings u, $u+1$ is u's parent, which is not a leaf. □

We call a hedge interval *complete* when if it contains a certain node, it also contains its children.

Observation 18. *In Algorithms 3 and 4, the following properties hold:*

(1) q_{until} is always a last sibling.
(2) q_{from} is always a leaf.
(3) $[q_{from}, q_{until}]$ is always a complete interval.

Proof. (1) In our initial call of TMatch, q_{until} is the root node of the tree, which is always a last sibling. The property for the deeper recursive calls follows immediately from a straightforward inspection of the recursive function calls in the algorithm.
(2) In our initial call of TMatch, q_{from} is the smallest node of Q, which is always a leaf. Furthermore, in TMatch we only call HMatch with q_{from} as a second parameter and TMatch with $q_{\text{best}}+1$ as a second parameter if q_{best} is not a last sibling (which is a leaf due to Observation 17). In HMatch all recursive calls have either q_{from} or rtop + 1 as second parameter. We show that, in this case, rtop is never a last sibling. Hence, according to Observation 17, rtop+1 is always a leaf. In the calls of TMatch on l.23, we have that rtop $< \infty$ and $q_{\text{hedge}} <$ rtop.lastSib, due to the while condition. As rtop $< \infty$, we have that rtop $\leq q_{\text{hedge}}$ due to the calls of RTop on l.21 and l.24. Hence, rtop $<$ rtop.lastSib. The proof is analogous for the calls of HMatch on l.31.
(3) In the initial call of TMatch, the claim obviously holds. In TMatch we call HMatch with q_{from} and q_{until}, for which the claim then trivially also holds; and TMatch with $q_{\text{best}}+1$ and q_{best}.lastSib if q_{best} is not a last sibling. Hence, $[q_{\text{best}}+1, q_{\text{best}}.\text{lastSib}]$ is the hedge subtree(q_{best}.nextSib) $\cdots$ subtree(q_{best}.lastSib), which is complete. The proof for the recursive calls in HMatch is analogous. □

Observation 19. *Let d_1 and d_2 be data nodes and q be a query node. If $[d_1, d_2]$ does not match subtree(q), then $[d_1, d_2]$ does not match any query tree interval containing subtree(q).*

Proof. Let q_{from} and q_{until} be such that $[q_{\text{from}}, q_{\text{until}}] = \text{subtree}(q)$. For $q'_{\text{from}} \leq q_{\text{from}}$ and $q'_{\text{until}} \geq q_{\text{until}}$, it can be shown by a simple structural induction on the hedge $[q'_{\text{from}}, q'_{\text{until}}]$ that $[d_1, d_2]$ does not match $[q'_{\text{from}}, q'_{\text{until}}]$. □

Observation 20. *Let H be a data hedge and $[q_{from}, q_{until}]$ be a complete query tree interval. We have that q is the largest node in $[q_{from}, q_{until}]$ such that $H \models [q_{from}, q]$ if and only if*

- *H matches $[q_{from}, q]$; and*
- *either $q = q_{until}$ or H does not match subtree$(q+1)$.*

Proof. Left to right: Let H be a data hedge and let $[q_{\text{from}}, q_{\text{until}}]$ be a query tree interval. Let q be the largest node in $[q_{\text{from}}, q_{\text{until}}]$ such that $H \models [q_{\text{from}}, q]$. If $q = q_{\text{until}}$ we are done. Otherwise, if, towards a contradiction, H matches subtree$(q+1)$, then we also immediately have that H matches $[q_{\text{from}}, q+1]$, which contradicts the maximality of q.

Right to left: Let q be a query node in $[q_{\text{from}}, q_{\text{until}}]$ such that H matches $[q_{\text{from}}, q]$. If $q = q_{\text{until}}$ then we are done. Otherwise, notice that, as $q+1$ is in the complete interval $[q_{\text{from}}, q_{\text{until}}]$, we have that subtree$(q+1)$ is entirely contained in $[q_{\text{from}}, q_{\text{until}}]$. Hence, if H does not match subtree$(q+1)$, then H also cannot match $[q_{\text{from}}, q+1]$. The latter can be shown by a simple structural induction on $[q_{\text{from}}, q+1]$. □

Correctness of TMatch. For readability, we split the correctness proof into several lemmas. Essentially, the proof is by induction on the height of the data node d in D.

Lemma 21. *Let d be a leaf data node and q_{from} and q_{until} be query nodes. Given d, q_{from}, and q_{until},* TMatch *is correct, that is,* TMatch *returns the largest node q in $[q_{from}, q_{until}]$ such that subtree$(d) \models [q_{from}, q]$ if it exists; and $q_{from} - 1$ otherwise.*

Proof. By induction on the number of nodes of $[q_{\text{from}}, q_{\text{until}}]$.

$q_{\text{from}} = q_{\text{until}}$: We initialize q_{best} with $q_{\text{from}} - 1$ on l.2. If d does not match q_{from} on l.5, we immediately return $q_{\text{best}} = q_{\text{from}} - 1$ on l.11. If d matches $q_{\text{from}} = q_{\text{until}}$ on l.5, q_{best} gets the value q_{from} on l.6. As $q_{\text{from}} = q_{\text{until}}$ is a last sibling (Observation 18), we do not execute the recursive call on l.8 and return q_{from} in l.9. Both cases are easily seen to be correct.

$q_{\text{from}} < q_{\text{until}}$: We initialize q_{best} with $q_{\text{from}} - 1$ on l.2. If d does not match q_{from} on l.5, we return $q_{\text{best}} = q_{\text{from}} - 1$ in l.11, which is correct. If d matches q_{from} in l.5, then q_{best} gets the value q_{from} and we enter the if-test on l.7. We need to consider two cases:

(1) *q_{from} is a last sibling:* In this case, we return q_{from} on l.9. This is correct, as $q_{\text{from}} + 1$ is q_{from}'s parent, which cannot be matched onto d due to the semantics of the descendant axis.

(2) *q_{from} is not a last sibling:* If q_{from} has a right sibling, we execute TMatch recursively on d, $q_{\text{from}} + 1$, and q_{from}.lastSib, yielding q. By induction, q

is computed correctly. That is, if $q = (q_{\text{from}} + 1) - 1$, which implies that d does not match $q_{\text{from}} + 1$, we return q_{from}, which is correct. Otherwise, we argue that subtree(d) = d matches $[q_{\text{from}}, q]$ but not subtree($q + 1$). By Observation 20, this would complete the proof. By induction, we immediately have that d matches $[q_{\text{from}}, q]$. If $q < q_{\text{from}}$.lastSib, we also have by induction that d does not match subtree($q+1$). If $q = q_{\text{from}}$.lastSib, then $q+1$ is q_{from}'s parent. Hence, d does not match subtree($q + 1$), as $q + 1$ has a child and d has not. □

Lemma 22. *Let d be a data node with height $n > 1$ and q_{from} and q_{until} be query nodes. If* HMatch *is correct for all data nodes of height up to $n - 1$, then* TMatch *is correct for all data nodes of height up to n. That is, given d, q_{from}, and q_{until},* TMatch *returns the largest node q in $[q_{from}, q_{until}]$ such that subtree(d) $\models [q_{from}, q]$ if it exists; and $q_{from} - 1$ otherwise.*

Proof. Assume that HMatch is correct for all data nodes of height up to $n-1$. As d is not a leaf, we start by calling HMatch on d.lastChild, q_{from}, and q_{until} on l.3 (see also Figure 3(a)), yielding q_{best}. By our assumption, q_{best} is computed correctly. We now prove the lemma by induction on the number of nodes of $[q_{\text{from}}, q_{\text{until}}]$.

$q_{\text{from}} = q_{\text{until}}$: We consider two cases.
(1) If subhedge(d.lastChild) does not match q_{from}, then q_{best} is $q_{\text{from}} - 1$. Consequently, we test whether d matches q_{from} on l.5. If d does not match q_{from}, we return $q_{\text{from}} - 1$ on l.11. If d matches q_{from}, then q_{best} gets the value q_{from}. As $q_{\text{from}} = q_{\text{until}}$ is a last sibling (Observation 18), we do not execute the recursive call on l.8 and return q_{from} in l.9. Both cases are easily seen to be correct.
(2) Otherwise, $q_{\text{best}} = q_{\text{from}} = q_{\text{until}}$. In this case we return q_{best}, which is correct.

$q_{\text{from}} < q_{\text{until}}$: (1) If both subhedge($d$.lastChild) and d do not match q_{from}, then we return $q_{\text{from}} - 1$ on l.11, which is correct.
(2) If subhedge(d.lastChild) matches q_{from} and $q_{\text{best}} = q_{\text{until}}$ on l.5, then we return q_{until}. Due to the correctness of HMatch, this means that subhedge(d.lastChild) already matches $[q_{\text{from}}, q_{\text{until}}]$, hence, subtree($d$) matches $[q_{\text{from}}, q_{\text{until}}]$ by our tree pattern matching semantics.
(3) If subhedge(d.lastChild) matches q_{from}, $q_{\text{best}} + 1 \leq q_{\text{until}}$, and d does not match $q_{\text{best}} + 1$ on l.5, then we return q_{best} in l.11. We consider two cases.

- *q_{best} is not a last sibling:* Hence, $q_{\text{best}} + 1$ is a leaf (Observation 17). Due to the correctness of HMatch for subhedge(d.lastChild), we know that subhedge(d.lastChild) does not match subtree($q_{\text{best}} + 1$) = $q_{\text{best}} + 1$. Hence, returning q_{best} is correct.
- *q_{best} is a last sibling:* Hence, $q_{\text{best}} + 1$ is q_{best}'s parent. Due to the correctness of HMatch, we have that subhedge(d.lastChild) $\models [q_{\text{from}}, q_{\text{best}}]$. Towards a contradiction, assume that subhedge(d) $\models$ subtree($q_{\text{best}} + 1$). As d does not match $q_{\text{best}} + 1$, this implies that subhedge(d.lastChild) $\models$ subtree($q_{\text{best}} + 1$). However, this contradicts that HMatch is correct. Hence, it is correct to return q_{best} due to Observation 20.

(4) Otherwise, denote by q^0_{best} the value of the variable q_{best} after the assignment on l.3. We have that q^0_{best} is correctly computed on l.3 and that d matches $q^0_{\text{best}} + 1$, after which q_{best} gets the value $q^0_{\text{best}} + 1$. Notice that $q^0_{\text{best}} + 1 \geq q_{\text{from}}$. We need to consider two cases:

- $q^0_{best} + 1$ *is a last sibling:* We return $q^0_{\text{best}} + 1$ in l.9. If $q^0_{\text{best}} + 1 = q_{\text{until}}$, this is correct. If $q^0_{\text{best}} + 1 < q_{\text{until}}$, towards a contradiction, assume that subtree(d) matches subtree(q^0_{best}+2). As q^0_{best}+2 is the parent of q^0_{best}+1, this would mean that subhedge(d.lastChild) $\models$ subtree($q^0_{\text{best}} + 1$), which is a contradiction.
- $q^0_{best} + 1$ *is not a last sibling:* If $q^0_{\text{best}} + 1$ has a right sibling, we execute TMatch on d, $q^0_{\text{best}} + 2$, and $q^0_{\text{best}} + 1$.lastSib on l.8, yielding q. By induction, q is computed correctly. If q is $(q^0_{\text{best}} + 2) - 1$, which implies that subtree(d) does not match $q^0_{\text{best}} + 2$, we return $q^0_{\text{best}} + 1$, which is correct. Otherwise, according to Observation 20, we need to show that subtree(d) matches $[q_{\text{from}}, q]$ but not subtree($q + 1$). By induction, we have that subtree(d) matches $[q_{\text{from}}, q]$. If $q < q^0_{\text{best}} + 1$.lastSib, we also have by induction that subtree(d) does not match subtree($q + 1$). If $q = q^0_{\text{best}} + 1$.lastSib, we have that subtree(d) does not match subtree($q + 1$), because there does not exist an $u \neq 1$ s.t. subtree(d) $\models$ subtree(q^0_{best}+1), and $q + 1$ is $q^0_{\text{best}} + 1$'s parent. □

Correctness of HMatch.

Lemma 23. *Let rtop* = RTop(q_1, q_2) *and* $q_1 \leq q_2$. *If* $q_1 \in$ *subhedge*(q_2), *then rtop* $= q_2$ *and* $q_2 \leq$ *rtop.lastSib. If* $q_1 \notin$ *subhedge*(q_2), *then rtop* $< q_2$ *and* $q_2 <$ *rtop.lastSib.*

Proof. Recall that, by definition, subhedge(q_2) is the interval $[q_{\text{small}}, q_2]$, where q_{small} is the smallest descendant of q_2's leftmost sibling.
$q_1 \in$ subhedge(q_2): As both q_1 and q_2 are in subhedge(q_2), we have that $[q_1, q_2]$ is entirely contained in subhedge(q_2).

By definition, rtop is the largest node in $[q_1, q_2]$ among the nodes with minimal depth. As q_2 has minimal depth in subhedge(q_2) and q_2 is the largest node in $[q_1, q_2]$, we have that rtop $= q_2$.
$q_1 \notin$ subhedge(q_2): Notice that this can only occur when q_2 has a parent. As $q_1 \leq q_2$, we have that $q_1 < q_{\text{small}}$. By definition of the left-to-right postordering, we have that q_1 is either a left sibling of an ancestor of q_2 (not including the ancestors themselves), or a descendant-or-self thereof. Let u_1 and u_2 be the two unique siblings such that $u_1 \neq u_2$, q_1 is in subtree(u_1), and q_2 is in subtree(u_2). Notice that $q_1 \leq u_1 < q_2 < u_2$. Hence, u_1 is in $[q_1, q_2]$ and depth(u_1) < depth(q_2). As q_2 has minimal depth in subhedge(q_2), we have that rtop is not in subhedge(q_2). By definition of RTop, this immediately implies that rtop $< q_2$. Furthermore, as depth(rtop.lastSib) = depth(rtop) $\leq$ depth(u_1) = depth(u_2) and as rtop.lastSib is also rtop's largest sibling, we have that rtop.lastSib $\geq u_2 > q_2$. □

Corollary 24. *If rtop* = RTop(q_1, q_2) *then* $q_1 \in$ *subhedge*(*rtop*).

Proof. As rtop is in $[q_1, q_2]$, rtop is also the rightmost node among the topmost nodes in $[q_1, \text{rtop}]$. If we assume that $q_1 \notin$ subhedge(rtop), then Lemma 23 implies that rtop $<$ rtop which is a contradiction. □

Lemma 25. *All function calls of* TMatch(d, q_1, q_2) *in the loop of* HMatch *have the property that* $[q_1, q_2]$ *is an interval which includes subtree*$(q_{hedge} + 1)$. *All function calls of* HMatch(d, q_1, q_2) *in the loop of* HMatch *have the property that* $[q_1, q_2]$ *is an interval which includes subtree*$(q_{tree} + 1)$.

Proof (Proof.). For the first statement, we have to show that *(i)* $q_1 \leq q_{\text{small}}$, where q_{small} is the smallest node in subtree$(q_{\text{hedge}} + 1)$ and *(ii)* $q_2 \geq q_{\text{hedge}} + 1$.

First, observe that the function calls of RTop on l.21 and l.24 results in a value of rtop that is at most q_{hedge}. If rtop $< q_{\text{hedge}}$ then rtop $< q_{\text{small}}$ as, by Lemma 23, rtop $= q_{\text{hedge}}$ when rtop is in $[q_{\text{small}}, q_{\text{hedge}}]$. Hence, rtop$+1 \leq q_{\text{small}}$. If rtop $= q_{\text{hedge}}$, we know that rtop is not a last sibling due to the condition of the while loop on l.22. Hence, $q_{\text{small}} = q_{\text{hedge}}+1 =$ rtop$+1$ is a leaf (Observation 17). This proves property *(i)*.

Property *(ii)* is immediate as the condition of the while-loop on l.22 requires that $q_2 =$ rtop.lastSib$\geq q_{\text{hedge}} + 1$.

The proof of the second statement is analogous to the proof of the first statement. □

Lemma 26. *The loop on line 18, and the while loops on lines 22 and 30 perform at most a linear number of iterations.*

Proof. Notice that we exit the loop on line 18 if $\max(q_{\text{tree}}, q_{\text{hedge}})$ does *not* increase. However, this value cannot keep increasing indefinitely as it is bounded from above by q_{until} in the algorithm. Hence, the loop performs at most a linear number of iterations.

The while loop on line 22 terminates after a linear number of iterations, as the value of rtop increases with each execution and the while loop only continues as long as rtop is smaller than q_{hedge}, a value which remains unchanged. The argument for the while loop on line 30 is analogous. □

Lemma 27. *Let* d *be a data node and* q_{from} *and* q_{until} *be query nodes. If* TMatch *is correct for all data nodes of height up to* n*, then* HMatch *is correct for all data nodes of height up to* n*. That is, given* d*,* q_{from}*, and* q_{until}*,* HMatch *returns the largest node* q *in* $[q_{from}, q_{until}]$ *such that subhedge*(d) *matches* $[q_{from}, q]$ *if it exists; and nil otherwise.*

Proof. Let k be such that d has k left siblings (including d itself). We prove the lemma by induction on k.

$\boxed{k = 1:}$ Immediate from the function call on l.14 and the assumption that TMatch is correct for all data nodes of height up to n.

$\boxed{k > 1:}$ We need to show that the algorithm returns $q_{\text{from}} - 1$ if subhedge(d) does not match q_{from}. Otherwise, we show that we return a q in $[q_{\text{from}}, q_{\text{until}}]$ if subhedge(d) matches $[q_{\text{from}}, q]$ and either

– $q = q_{\text{until}}$, or
– neither subtree(d), nor subhedge(d.prevSib) matches subtree($q+1$).

In the remainder of the proof, we refer to the above property with the label (†). The correctness of property (†) follows directly from our tree pattern query semantics if we return $q_{\text{from}} - 1$ and from Observation 20 otherwise. Indeed, from Observation 18 we know that $[q_{\text{from}}, q_{\text{until}}]$ is complete. Furthermore, subhedge(d) does not match subtree($q+1$) if and only if neither subtree(d) nor subhedge(d.prevSib) match subtree($q+1$).

Notice that the loop on l.18 terminates by Lemma 26. We now proceed with an induction over the number ℓ of loop executions proving that the following invariants hold:

(I1): if q_{tree} is not $q_{\text{from}} - 1$ then subhedge(d) matches $[q_{\text{from}}, q_{\text{tree}}]$;
(I2): if q_{hedge} is not $q_{\text{from}} - 1$ then subhedge(d) matches $[q_{\text{from}}, q_{\text{hedge}}]$;
(I3): $q_{\text{tree}} = q_{\text{until}}$ or subtree(d) does not match subtree($q_{\text{tree}}+1$); and,
(I4): $q_{\text{hedge}} = q_{\text{until}}$ or subhedge(d.prevSib) does not match subtree($q_{\text{hedge}}+1$).

At the same time, we show that, if the algorithm returns a certain value q, the property (†) holds for q.

$\ell = 0$ (before the first loop execution): We computed q_{hedge}, which results from executing HMatch on d.prevSib, q_{from}, and q_{until}; and we computed q_{tree}, which results from executing TMatch on d, q_{from}, and q_{until} (see also Figure 3(c)). By induction on k, we have that q_{hedge} is computed correctly. Moreover, as we assume that TMatch is correct for all data nodes of height up to n, we also have that q_{tree} is computed correctly. Properties (I1)–(I2) immediately follow from the correctness of the recursive calls of TMatch and HMatch. Moreover, Observation 20 implies that (I3)–(I4) also hold. As the algorithm does not return anything up to here, we do not have to show yet that (†) holds.

$\ell \geq 1$ (subsequent loop executions): We consider three cases.
(1) If $q_{\text{hedge}} = q_{\text{tree}}$, we return q_{hedge}. This is correct, as in this case, properties (I1)–(I4) immediately imply property (†).
(2) If $q_{\text{tree}} < q_{\text{hedge}}$, notice that we do not change the value of q_{hedge} in this iteration of the loop. Hence, for the induction, we only need to show that properties (I1) and (I3) are preserved. We consider two cases.
If $q_{\text{hedge}} = q_{\text{until}}$ the while loop in l.22 is not executed and we return q_{until} in l.26. Here, it follows immediately from (I2) that (†) holds.
If $q_{\text{hedge}} < q_{\text{until}}$ we consider two cases.

– If subtree(d) does not match subtree($q_{\text{hedge}}+1$), none of the function calls TMatch(d, q_1, q_2) in the while loop yield a value greater than q_{hedge}. This follows from the correctness of TMatch for data nodes up to height n, and from Lemma 25, stating that $[q_1, q_2]$ always includes subtree($q_{\text{hedge}}+1$). Indeed, should such a function call TMatch(d, q_1, q_2) yield a greater value than q_{hedge}, then we would have that subtree(d) matches subtree($q_{\text{hedge}}+1$), which contradicts that we are investigating the case that subtree(d) does not match subtree($q_{\text{hedge}}+1$). Hence, we return q_{hedge} in line 26. Correctness of the property (†) for q_{hedge} now follows from the following facts:

- $q_{\text{hedge}} \geq q_{\text{from}}$, as $q_{\text{tree}} < q_{\text{hedge}}$;
- $q_{\text{hedge}} < q_{\text{until}}$;
- subhedge(d) matches $[q_{\text{from}}, q_{\text{hedge}}]$, by (I2);
- subtree(d) does not match subtree($q_{\text{hedge}} + 1$); and,
- subhedge(d.prevSib) does not match subtree($q_{\text{hedge}} + 1$) by (I4).

– If subtree(d) matches subtree($q_{\text{hedge}}+1$) the proof is more complicated. First, observe that the while loop on l.22 terminates by Lemma 26.
For the remainder of this case, we will show that $q_{\text{tree}} > q_{\text{hedge}}$ after exiting the while loop in the $i + 1^{\text{th}}$ execution of the test on l.22. In particular, this implies that the algorithm will not return any value in iteration ℓ of the loop. So we only need to show that, at the end of the current iteration, properties (I1) and (I3) hold.
To show (I3), we will show that, if in the j^{th} execution of the while loop we obtain a value q for the variable q_{tree} for which it holds that $q > q_{\text{hedge}}$ then we either have that $q = q_{\text{until}}$ or that subtree(d) does not match subtree($q + 1$). Afterwards, we show (I1).
We start by showing that $q_{\text{tree}} > q_{\text{hedge}}$ after exiting the while loop:
Goal 1: *$q_{tree} > q_{hedge}$ after exiting the while loop in the $i + 1^{th}$ execution of the test on l.22. So we execute the while body i times and then exit the loop.*
Let q^i_{tree} denote the value of q_{tree} at the end of the i^{th} execution (i.e., after the assigment on l.23) and let q^0_{tree} be the value of q_{tree} before entering the while loop. Furthermore, let rtop^i denote the value of rtop at the end of the i^{th} execution (i.e., after the assigment on l.24). Let rtop^0 be the value of rtop before entering the while loop.
$\boxed{i = 0:}$ We will show that this case does not occur. That is, the body of the while loop is always executed at least once. Towards a contradiction, assume that we do not execute the body of the while loop. We consider two cases. If we exit the while loop one of them must hold.

- *Case 1:* $\text{rtop}^0 < \infty$ and $q_{\text{hedge}} \geq \text{rtop}^0.\text{lastSib}$. Recall that $\text{rtop}^0 = \text{RTop}(q^0_{\text{tree}} + 1, q_{\text{hedge}})$. Due to Lemma 23, $q_{\text{hedge}} \geq \text{rtop}^0.\text{lastSib}$ implies that *(i)* $\text{rtop}^0 = \text{rtop}^0.\text{lastSib} = q_{\text{hedge}}$ and that *(ii)* $q^0_{\text{tree}} + 1$ is in subhedge(q_{hedge}). As $q_{\text{hedge}} < q_{\text{until}}$ and q_{hedge} is a last sibling this means that $q^0_{\text{tree}}+1$ is in subtree($q_{\text{hedge}}+1$). Moreover, as we are in the case that $q_{\text{tree}} < q_{\text{hedge}}$, we know by induction on ℓ (statement (I3) in particular) that subtree(d) does not match subtree($q^0_{\text{tree}} + 1$). However, as we have shown above that $q^0_{\text{tree}} + 1$ is in subtree($q_{\text{hedge}} + 1$), this contradicts the fact that we are in the case that subtree(d) matches subtree($q_{\text{hedge}} + 1$).
- *Case 2:* $\text{rtop}^0 = \infty$. By definition of RTop, this means that $q^0_{\text{tree}} + 1 > q_{\text{hedge}}$. But we are currently investigating in the case that $q^0_{\text{tree}} < q_{\text{hedge}}$. Contradiction.

Hence, we showed that the while loop on l.22 is executed at least once.
$\boxed{i > 0:}$ Again, we consider the two possible settings in which we exit the while loop. We show again that the first of the two does not occur here.

- *Case 1:* $\text{rtop}^i < \infty$ and $q_{\text{hedge}} \geq \text{rtop}^i.\text{lastSib}$. Recall that $\text{rtop}^i = \text{RTop}(q^i_{\text{tree}} + 1, q_{\text{hedge}})$. Due to Lemma 23, $q_{\text{hedge}} \geq \text{rtop}^i.\text{lastSib}$ implies that *(i)* $\text{rtop}^i = \text{rtop}^i.\text{lastSib} = q_{\text{hedge}}$ and that *(ii)* $q^i_{\text{tree}} + 1$ is in

subhedge(q_{hedge}), implying that $q^i_{\text{tree}} + 1 \leq q_{\text{hedge}}$. As $q_{\text{hedge}} < q_{\text{until}}$ and q_{hedge} is a last sibling this means that $q^i_{\text{tree}} + 1$ is in subtree($q_{\text{hedge}} + 1$). Since we did not exit the while loop in the ith test, we have that $q_{\text{hedge}} < \text{rtop}^{i-1}.\text{lastSib}$. Hence, we have that $q^i_{\text{tree}} + 1 \leq q_{\text{hedge}} < \text{rtop}^{i-1}.\text{lastSib}$. Recall that $q^i_{\text{tree}} = \text{HMATCH}(d.\text{prevSib}, \text{rtop}^{i-1} + 1, \text{rtop}^{i-1}.\text{lastSib})$. By the correctness of TMATCH, Observation 20, and the fact that $[\text{rtop}^{i-1} + 1, \text{rtop}^{i-1}.\text{lastSib}]$ is a complete interval (Observation 18) we can conclude that subtree(d) does not match subtree($q^i_{\text{tree}} + 1$) which, we argued above, is a subtree of subtree($q_{\text{hedge}} + 1$). Hence, subtree(d) does not match subtree($q_{\text{hedge}} + 1$), which contradicts the fact that we are in the case that subtree(d) matches subtree($q_{\text{hedge}} + 1$).

- *Case 2:* $\text{rtop}^i = \infty$. Hence, $q^i_{\text{tree}} + 1 > q_{\text{hedge}}$. We prove that it cannot be the case that $q^i_{\text{tree}} = q_{\text{hedge}}$. Hence, $q^i_{\text{tree}} > q_{\text{hedge}}$ and Goal 1 follows. To this end, assume, towards a contradiction, that $q^i_{\text{tree}} = q_{\text{hedge}}$. Recall that $q^i_{\text{tree}} = \text{TMATCH}(d, \text{rtop}^{i-1} + 1, \text{rtop.lastSib})$. Moreover, $\text{rtop}^{i-1}.\text{lastSib} > q_{\text{hedge}}$ since otherwise we would have exited the while loop right after test i. We conclude that $q_{\text{hedge}} + 1$ is a node in $[\text{rtop}^{i-1}+1, \text{rtop}^{i-1}.\text{lastSib}]$. However, as subtree($d$) matches subtree($q_{\text{hedge}}+1$), this would imply that subtree(d) also matches $[\text{rtop}^{i-1} + 1, q_{\text{hedge}} + 1] = [\text{rtop}^{i-1} + 1, q^i_{\text{tree}} + 1]$ which is in contradiction with the correctness of TMATCH.

This concludes the proof of Goal 1.

Goal 2. *If in the j^{th} execution of the while loop we obtain a value q for the variable q_{tree} for which it holds that $q > q_{hedge}$ and $q + 1 \leq q_{until}$, then we have that subtree(d) does not match subtree($q + 1$).*

Observe that we need at least one execution of the body of the while, since before the first execution we have that $q_{\text{tree}} < q_{\text{hedge}}$. Let q^j_{tree} denote the value of q_{tree} at the end of the j^{th} execution (i.e., after the assigment on l.23) and let q^0_{tree} be the value of q_{tree} before entering the while loop. Furthermore let rtop^j denote the value of rtop at the end of the j^{th} execution (i.e., after the assigment on l.24). Let rtop^0 be the value of rtop before entering the while loop.

Hence, for every $j \geq 1$, q^j_{tree} is the result of a function call $\text{TMATCH}(d, \text{rtop}^{j-1}+1, \text{rtop}^{j-1}.\text{lastSib})$. If $q^j_{\text{tree}} > q_{\text{hedge}}$ we will exit the while loop right after the current iteration. We consider three cases.

- If $q^j_{\text{tree}} < \text{rtop}^{j-1}.\text{lastSib}$ we have that subtree(d) does not match the subtree of $q^j_{\text{tree}} + 1$ due to the correctness of TMATCH for data nodes up to height n and Observation 20.
- If $q^j_{\text{tree}} = q_{\text{until}}$ the claim is trivial.
- The remaining case is that $q^j_{\text{tree}} = \text{rtop}^{j-1}.\text{lastSib} < q_{\text{until}}$. In this case, $q^j_{\text{tree}} + 1$ is the parent of q^j_{tree} due to Observation 17. We consider two cases.

  $\boxed{j = 1:}$ We want to prove that subtree(d) does not match subtree($q^0_{\text{tree}} + 1$) and that subtree($q^0_{\text{tree}} + 1$) is a subtree of subtree($q^1_{\text{tree}} + 1$). Then we can conclude that subtree(d) does not match subtree($q^1_{\text{tree}} + 1$).

  We start by proving that subtree(d) does not match subtree($q^0_{\text{tree}}+1$). By induction on ℓ (and, in particular, by (I3)) we know that $q^0_{\text{tree}} = q_{\text{until}}$ or

subtree(d) does not match subtree($q^0_{\text{tree}}+1$). If $q^0_{\text{tree}} = q_{\text{until}}$ we wouldn't be in the case that $q^0_{\text{tree}} < q_{\text{hedge}}$. We can conclude that subtree(d) does not match subtree($q^0_{\text{tree}}+1$) .

It remains to be shown that subtree($q^0_{\text{tree}}+1$) is a subtree of subtree($q^1_{\text{tree}}+1$). Line 21 states that $\text{rtop}^0 = \text{RTop}(q^0_{\text{tree}}+1, q_{\text{hedge}})$. Corollary 24 implies that then $q^0_{\text{tree}}+1$ is a node in subhedge(rtop^0). Now we take into consideration that we are investigating in the case that $q^1_{\text{tree}} = \text{rtop}^0.\text{lastSib}$ which implies that subhedge(rtop^0) $\subseteq$ subhedge(q^1_{tree}). Combining this with the consequence of the Corollary it follows that $q^0_{\text{tree}}+1$ is a node in subhedge(q^1_{tree}). Recall that $q^1_{\text{tree}}+1$ is q^1_{tree}'s parent. Hence, $q^0_{\text{tree}}+1$ is a node in subtree($q^1_{\text{tree}}+1$) and subtree($q^0_{\text{tree}}+1$) is a subtree of subtree($q^1_{\text{tree}}+1$).

$\boxed{j > 1:}$ Analogously as in the $j = 1$ case, we prove that subtree(d) does not match subtree($q^{j-1}_{\text{tree}}+1$) and that subtree($q^{j-1}_{\text{tree}}+1$) is a subtree of subtree($q^j_{\text{tree}}+1$). Then we conclude that subtree(d) does not match subtree($q^j_{\text{tree}}+1$).

We start by proving that subtree(d) does not match subtree($q^{j-1}_{\text{tree}}+1$). We have that $q^{j-1}_{\text{tree}} = \text{TMatch}(d, \text{rtop}^{j-2}+1, \text{rtop}^{j-2}.\text{lastSib})$. Notice that, if $q^{j-1}_{\text{tree}} < \text{rtop}^{j-2}.\text{lastSib}$, we immediately have by the correctness of TMatch and Observation 20 that subtree(d) does not match subtree($q^{j-1}_{\text{tree}}+1$). So, towards a contradiction, assume that $q^{j-1}_{\text{tree}} \geq \text{rtop}^{j-2}.\text{lastSib}$.

Notice that $q_{\text{hedge}} < \text{rtop}^{j-2}.\text{lastSib}$ and that $\text{rtop}^{j-1} \leq q_{\text{hedge}}$, otherwise we wouldn't have arrived in the jth iteration. Moreover, $\text{rtop}^{j-2} < \text{rtop}^{j-1}$. As $\text{rtop}^{j-2} < \text{rtop}^{j-1} \leq \text{rtop}^{j-2}.\text{lastSib}$, we also have that $\text{rtop}^{j-1}.\text{lastSib} \leq \text{rtop}^{j-2}.\text{lastSib}$. This implies that $\text{rtop}^{j-1}.\text{lastSib} \leq \text{rtop}^{j-2}.\text{lastSib} < q^{j-1}_{\text{tree}}+1 \leq q^j_{\text{tree}}$ which contradicts that $q^j_{\text{tree}} = \text{rtop}^{j-1}.\text{lastSib}$, which is the case we are investigating.

It remains to be shown that subtree($q^{j-1}_{\text{tree}}+1$) is a subtree of subtree($q^j_{\text{tree}}+1$). Line 24 states that $\text{rtop}^{j-1} = \text{RTop}(q^{j-1}_{\text{tree}}+1, q_{\text{hedge}})$. Corollary 24 implies that then $q^{j-1}_{\text{tree}}+1$ is a node in subhedge(rtop^{j-1}). Now we take into consideration that we are investigating the case that $q^j_{\text{tree}} = \text{rtop}^{j-1}.\text{lastSib}$ which implies that subhedge(rtop^{j-1}) $\subseteq$ subhedge(q^j_{tree}). Combining this with the consequence of the Corollary it follows that $q^{j-1}_{\text{tree}}+1$ is a node in subhedge(q^j_{tree}). Recall that $q^j_{\text{tree}}+1$ is q^j_{tree}'s parent. Hence, $q^{j-1}_{\text{tree}}+1$ is a node in subtree($q^j_{\text{tree}}+1$) and subtree($q^{j-1}_{\text{tree}}+1$) is a subtree of subtree($q^j_{\text{tree}}+1$).

This concludes the proof of Goal 2.

It remains to show that (I1) holds at the end of the ℓ-th iteration of the loop, that is, that subhedge(d) matches $[q_{\text{from}}, q_{\text{tree}}]$. Due to (I2) we have that subhedge(d) matches $[q_{\text{from}}, q_{\text{hedge}}]$. Recall that the number of while loop executions is at least one. Hence, we have that $q_{\text{tree}} = \text{TMatch}(d, \text{rtop}+1, \text{rtop.lastSib})$, where $\text{rtop} \leq q_{\text{hedge}} < q_{\text{tree}} \leq \text{rtop.lastSib}$. The first inequality follows from the fact that $\text{rtop} < \infty$ and the definition of RTop, the second one follows from Goal 1, and the third one from the correctness of TMatch. Hence, we have that

- subhedge(d) $\models [q_{\text{from}}, \text{rtop}]$ and
- subtree(d) $\models [\text{rtop}+1, q_{\text{tree}}]$.

Moreover, the facts that rtop + 1 is a leaf (Observation 17) and $q_{\text{tree}} \leq$ rtop.lastSib imply that subhedge(d) $\models [q_{\text{from}}, q_{\text{tree}}]$.
This concludes the proof the case where subtree(d) matches subtree($q_{\text{hedge}}+1$).

This concludes the proof of the case where $q_{\text{hedge}} < q_{\text{until}}$, and also the proof of the case where $q_{\text{tree}} < q_{\text{hedge}}$.
(3) If $q_{\text{hedge}} < q_{\text{tree}}$ the proof is dual to the proof of case (2). □

The correctness of Lemma 6 now follows from Lemmas 21, 22, and 27.

Proof of Theorem 7. TMatch-All *is correct, that is,* TMatch-All(D, Q) *outputs the data nodes u such that $D \models^u Q$.*

Proof. Immediate from the correctness of TMatch, that is, Lemma 6. □

Complexity Proof of Lemma 8. *Given the root node of a data tree D, and the smallest and largest query nodes and of a query tree Q, respectively,* TMatch *runs in time $\mathcal{O}(|D||Q|depth(Q))$. Moreover,* TMatch *performs at most $|D||Q|$ data versus query node comparisons.*

Proof (Proof sketch.). Let $|D|$ and $|Q|$ be the number of nodes in the data and query tree, respectively. We first show by induction on the height n of the data node d that the number of calls to the function TMatch in the computation tree is at most $|D||Q|$. To this end, we prove three intermediate goals.

Goal 1: *Let d be a leaf data node. A computation of* TMatch(d, q_{from}, q_{until}) *yielding result q makes at most $|[q_{from}, q+1]|$ calls to* TMatch.

By induction on the size of the query tree interval $[q_{\text{from}}, q_{\text{until}}]$. If d is a leaf and $q_{\text{from}} = q_{\text{until}}$, then TMatch does not call HMatch recursively and the test on l.7 fails. Therefore, there is only 1 call to TMatch and the induction hypothesis holds. If $q_{\text{from}} < q_{\text{until}}$, and TMatch is not called recursively, then the minimal value we return is $q_{\text{from}} - 1$. Again, there is only 1 call to TMatch and the induction hypothesis holds. Otherwise, we call TMatch on l.8, yielding result q. By induction, the total number of calls to TMatch is at most $1 + |[q_{\text{from}}+1, q+1]|$. As $|[q_{\text{from}}, q+1]| = 1 + |[q_{\text{from}}+1, q+1]|$, the induction holds. This concludes the proof of Goal 1.

Goal 2: *Let d be a data node with height $n > 1$. If the computation of* HMatch$(d.lastChild, q_{from}, q_{until})$*, yielding result q^0_{best}, performs at most $|subhedge(d.lastChild)| \cdot |[q_{from}, q^0_{best}+1]|$ calls to* TMatch*, then the computation of* TMatch(d, q_{from}, q_{until})*, yielding result q, makes at most $|subtree(d)| \cdot |[q_{from}, q+1]|$ calls to* TMatch.

We prove Goal 2 by induction on the size of the query tree interval $[q_{\text{from}}, q_{\text{until}}]$. TMatch starts by calling HMatch$(d.\text{lastChild}, q_{\text{from}}, q_{\text{until}})$ yielding q^0_{best}. Hence, $|\text{subhedge}(d.\text{lastChild})| \cdot |[q_{\text{from}}, q^0_{\text{best}}+1]|$ calls to TMatch are performed by this subroutine.

If $q_{\text{from}} = q_{\text{until}}$, then we either return q^0_{best} on l.11 or $q^0_{\text{best}}+1$ on l.9. In both cases, the total number of calls to TMatch is at most $1+|\text{subhedge}(d.\text{lastChild})| \cdot |[q_{\text{from}}, q^0_{\text{best}}+1]|$ which is at most $|\text{subtree}(d)| \cdot |[q_{\text{from}}, q^0_{\text{best}}+1]|$.

If $q_{\text{from}} < q_{\text{until}}$, and TMATCH is not called recursively, then the minimal value we return is q^0_{best}. Again, the number of calls to TMATCH is at most $1+|\text{subhedge}(d.\text{lastChild})| \cdot |[q_{\text{from}}, q^0_{\text{best}}+1]|$ and the induction hypothesis holds. Otherwise, we call TMATCH on l.8, yielding result q. By induction, the total number of calls to TMATCH is at most $1+|\text{subhedge}(d.\text{lastChild})| \cdot |[q_{\text{from}}, q^0_{\text{best}}+1]| + |\text{subtree}(d)| \cdot |[q^0_{\text{best}}+2, q+1]|$ which is at most $|\text{subtree}(d)| \cdot |[q_{\text{from}}, q+1]|$. This concludes the proof of Goal 2.

Goal 3: *Let d be a data node. If the computation of* $\text{TMATCH}(d, q_1, q_2)$*, yielding q_{tree} makes at most $|subtree(d)| \cdot |[q_1, q_{tree}+1]|$ calls to* TMATCH*, then the computation of* $\text{HMATCH}(d, q_{from}, q_{until})$*, yielding q makes at most $|subhedge(d)| \cdot |[q_{from}, q+1]|$ calls to* TMATCH*.*

Let k be such that d has k left siblings (including d itself). We prove the lemma by induction on k. If $k = 1$, Goal 3 is an immediate consequence from the assumption of Goal 3 and the recursive call of TMATCH on l.14. If $k > 1$, then we start by calling $\text{HMATCH}(d.\text{prevSib}, q_{\text{from}}, q_{\text{until}})$, yielding $q^{1,0}_{\text{hedge}}$, and calling $\text{TMATCH}(d, q_{\text{from}}, q_{\text{until}})$, yielding $q^{1,0}_{\text{tree}}$. By induction on k, we have that the call of HMATCH induces $|\text{subhedge}(d.\text{prevSib})| \cdot |[q_{\text{from}}, q^{1,0}_{\text{hedge}}+1]|$ calls to TMATCH. Moreover, by the statement of Goal 3, we have that the recursive call of TMATCH induces $|\text{subtree}(d)| \cdot |[q_{\text{from}}, q^{1,0}_{\text{tree}}+1]|$ calls to TMATCH in total.

According to Lemma 26, the loops on l.18, 22, and 30 perform at most a linear number of iterations. Hence, TMATCH and HMATCH are called (directly) at most a quadratic number of times in the loop.

By $q^{i,j}_{\text{tree}}$, we denote the value of the variable q_{tree} in the i-th iteration of the loop and at the end of the j-th iteration of the while loop in l.22. Moreover, let ℓ denote the number of loop executions and let $\max_i$ denote the number of executions of the while loop on l.22 in the i^{th} loop execution. Then, we have that every computation of $\text{TMATCH}(d, q_1, q_2)$ in the while loop performs at most $|\text{subtree}(d)| \cdot |[q^{i,j-1}_{\text{tree}}+2, q^{i,j}_{\text{tree}}+1]|$ calls to TMATCH when $j > 1$ and at most $|\text{subtree}(d)| \cdot |[q^{i-1,\max_{i-2}(j)}_{\text{tree}}+2, q^{i,1}_{\text{tree}}+1]|$ calls otherwise. Notice that $q^{1,0}_{\text{tree}} < q^{1,1}_{\text{tree}} < \cdots < q^{1,\max_1}_{\text{tree}} < q^{2,1}_{\text{tree}} < \cdots < q^{\ell,\max_\ell}_{\text{tree}} \leq q$, where q is the value we return. Hence, the sum of the calls to TMATCH made by the computations of TMATCH on l.23 is at most $|\text{subtree}(d)| \cdot |[q^{1,0}_{\text{tree}}+2, q+1]|$.

Analogously, we obtain that the sum of the calls to TMATCH by the computations of HMATCH on l.31 is at most $|\text{subhedge}(d.\text{prevSib})| \cdot |[q^{1,0}_{\text{tree}}+2, q+1]|$.

In total, this means that the number of calls to TMATCH is at most $|\text{subhedge}(d.\text{prevSib})| \cdot |[q_{\text{from}}, q^{1,0}_{\text{hedge}}+1]| + |\text{subhedge}(d.\text{prevSib})| \cdot |[q^{1,0}_{\text{hedge}}+2, q+1]| + |\text{subtree}(d)| \cdot |[q_{\text{from}}, q^{1,0}_{\text{tree}}+1]| + |\text{subtree}(d)| \cdot |[q^{1,0}_{\text{tree}}+2, q+1]|$ which is at most $|\text{subhedge}(d)| \cdot |[q_{\text{from}}, q+1]|$. Hence, Goal 3 follows.

As a consequence of Goals 1, 2, and 3, the total number of calls to TMATCH performed by the algorithm is $|D||Q|$. As the only data versus query node comparison in the algorithm occurs in l.5 of TMATCH, and as each call of TMATCH performs at most one data versus query node comparison (excluding comparisons in recursive calls), the total algorithm also performs at most $|D||Q|$ data versus query node comparisons.

We now argue how this leads us to showing that the overall algorithm has polynomial running time. Consider the entire tree of the calls to TMatch and HMatch in the algorithm, where the children of a node are the functions it calls directly. This computation tree contains at most $|D||Q|$ calls of TMatch. Moreover, every call of HMatch performs at least one direct recursive call to TMatch, so the computation tree also contains at most $|D||Q|$ calls of HMatch. Analogously, the entire computation tree contains at most $|D||Q|$ calls to rtop. As rtop can be implemented to run in time $\mathcal{O}(\text{depth}(Q))$, the total algorithm runs in time $\mathcal{O}(|D||Q|\text{depth}(Q))$. □