Characterizing XML Twig Queries with Examples

Sławek Staworko[1] and Piotr Wieczorek[2]

1 **University of Lille 3, INRIA LINKS, CNRS**
Lille, France
`slawomir.staworko@inria.fr`

2 **University of Wrocław, Institute of Computer Science**
Wrocław, Poland
`piotrek@cs.uni.wroc.pl`

Abstract

Typically, a (Boolean) query is a finite formula that defines a possibly infinite set of database instances that satisfy it (*positive* examples), and implicitly, the set of instances that do not satisfy the query (*negative* examples). We investigate the following natural question: for a given class of queries, is it possible to *characterize* every query with a finite set of positive and negative examples that no other query is consistent with.

We study this question for *twig* queries and XML databases. We show that while twig queries are characterizable, they generally require exponential sets of examples. Consequently, we focus on a practical subclass of *anchored* twig queries and show that not only are they characterizable but also with polynomially-sized sets of examples. This result is obtained with the use of *generalization operations* on twig queries, whose application to an anchored twig query yields a properly contained and minimally different query. Our results illustrate further interesting and strong connections between the structure and the semantics of anchored twig queries that the class of arbitrary twig queries does not enjoy. Finally, we show that the class of unions of twig queries is not characterizable.

1998 ACM Subject Classification H.2.3 Query languages, H.2.1 Normal forms

Keywords and phrases Query characterization, Query examples, Query fitting, Twig queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2015.144

1 Introduction

One of the central, if not defining, instruments in computer science is using a formula, a finite syntactic object, to define a (possibly infinite) set of its models. A typical example are regular expressions that define languages of words. Database queries also fall into this category, which is best illustrated with Boolean queries: a query q defines the set of instances satisfying q, *positive examples*, and implicitly the set of instances that do not satisfy q, *negative examples*. In this paper, we study the question of (finite) *characterizability*: Can every query be characterized with a finite set of examples? More precisely, given a class of queries $\mathcal{Q}$ is it possible for every $q \in \mathcal{Q}$ to find a set of examples such that q is the only query (modulo equivalence) in $\mathcal{Q}$ *consistent* with it i.e., a query satisfying all positive examples and none of negative examples. And if it is possible, can we say anything about the number and the sizes of the necessary examples?

The question of characterizability arises naturally in the context of learning/teaching [9, 6] which deals with the problem of constructing a formula (query) consistent with a given set of examples. However, research on characterizability has a number of potential applications of independent interest because it yields way to generate a set of examples consistent with a

© Sławek Staworko and Piotr Wieczorek;
licensed under Creative Commons License CC-BY

given query. Such examples can be used, for instance, for elementary query engine debugging, and query visualization and explanation. The exhaustive nature of the examples provided by characterizability i.e., there is exactly one query satisfying the examples, may also be useful in final, verification, stages of reverse engineering of database queries.

In this paper, we investigate the problem of characterizability for XML databases and twig queries [2, 14]. XML documents can be seen as labeled unranked trees and twig queries are tree-shaped patterns that additionally use a wildcard label $\star$ (matching any label) and descendant edges (matching a path of positive length). Twig queries are the core of virtually any XML query language and have been extensively studied in the literature [4]. In particular, learning twig queries from examples has been previously investigated [19, 5] and the current paper can be seen as a continuation of this line of work and an attempt at deepening our understanding of the relationship between a query and its examples.

In essence, our results show that characterizability is a measure of richness of the query class, which is closely related to its expressive power. We show that unions of twig queries are not characterizable because virtually any set of examples has infinitely many consistent queries i.e., the class of unions of twig queries is very rich. On the other hand, the class of twig queries is less rich, given a tree the number of twig queries satisfied in it is finite, which allows to show that twig queries are characterizable.

While twig queries are characterizable with finite sets of examples, we show that the number of necessary examples may be exponential. The main contribution of this paper is showing that (polynomially) small sets of examples are sufficient to characterize *anchored twig queries* [19]. In essence, this subclass of twig queries forbids descendant edges incident to a $\star$ node, which merely prevents from imposing conditions on ancestry with a lower bound on depth: "a node x is a ancestor of node y and the path from x to y is of length $\geq k$", where $k > 1$. While the expressive power does not seem to be significantly restricted from the practical point of view, this class of twig queries exhibits a very close relationship between the structure and the semantics: containment of two twig queries is equivalent to the existence of an embedding between the queries, an equivalence that does not hold for arbitrary twig queries [13]. This relationship between the structure of the query and its semantics goes even further [19]: the use of two positive examples of an anchored query q allows to identify all queries that contain q. We continue to explore this relationship and deepen its understanding by characterizing the structure of the semi-lattice of anchored twig queries: for a given anchored twig query q we are interested in the number and the sizes of the most specific anchored twig queries properly containing q, and interestingly, we show that their number and their sizes are polynomially small (w.r.t. the size of q). This result is essential in showing that anchored twig queries have polynomially-sized characterizing sets of examples.

To understand the existence of an embedding, and hence the containment, between two anchored twig queries, we study three generalization operations applied to twig queries (cf. Fig. 5): 1) changing a label to $\star$, 2) changing the type of an edge from child to descendant, and 3) removing a node. While natural and elementary, these operations capture precisely the subclass of *injective* (*ancestor-preserving*) embeddings between queries: query p can be obtained from a query q by applying a sequence of generalization operations if and only if there is an injective embedding of q into p. Consequently, when applied, in a diligent manner, to anchored twig queries they allow to characterize the semi-lattice of anchored twig queries under injective semantics. We also show that anchored twig queries have unique canonical forms which can be efficiently obtained by iteratively applying the operations and the unique canonical form is in fact the (size-)minimal equivalent query. This further illustrates the desirable properties of anchored twig queries as minimization for arbitrary twig queries is known to be intractable [2, 13]. We point out, however, that classes of twig queries properly containing anchored twig queries are known to have tractable minimization based on operations reducing the query [11].

To extend the characterization results from injective to standard semantics, we identify mapping overlap as the essential difference between the corresponding two type of embeddings: unlike the injective embedding, the standard embedding of a query q into a query p may map different fragments of q into the same fragment of p creating an overlap of the images of the different fragments of q. To address this difference we explore using a duplication operation that creates separate copies of the fragment of p thus eliminating the overlap. However, this operation introduces redundancies and may increase arbitrarily the size of the query. Consequently, it is applied together with different generalization operations to avoid introducing redundancies and to avoid undesirable growth of the query we devise a recursive pattern of applying duplication operations that allows to polynomially bound the number of introduced copies. The number of the generalizations is linear in the size of the original query, and thanks to applying the duplication operation in a controlled manner, the size of each generalization is at most quadratic. The generalizations are then used to construct a set of characterizing examples consisting of a polynomial number of examples each of polynomially bounded size.

The main contributions of the paper are:

- We formulate the problem of characterizability of a Boolean class of XML queries with examples and study it for classes of twig queries.
- We show that unions of twig queries are not characterizable while twig queries alone are but the number of examples necessary to characterize a twig query may be exponential in the size of the query.
- We investigate characterizability of a rich subclass of anchored twig queries, and propose a set of natural generalization operations that allows to characterize with a polynomially-sized set of examples any anchored twig query under the injective (ancestor-preserving) semantics.
- We propose a duplication operation whose diligent use allows to extend the characterizability to anchored twig queries to the standard semantics.

Related work. Our work is closely related to the scenario of *teaching* [10, 9, 16] where the set of examples to be presented is selected by a helpful teacher. Goldman and Kearns [9] define a sequence of positive and negative examples to be a *teaching sequence* for a given concept c if it uniquely specifies c in the concept class C. Hence, this is essentially the same idea as in our case. They study the properties of a *teaching dimension* of a concept class that is a minimum number of examples a teacher has to reveal in order to uniquely identify any concept in the class. They consider, however, different concept classes then ours, namely orthogonal rectangles and boolean formulas. Also, teaching sequences for other classes of boolean formulas has been studied in [18, 3].

Recently, learning and verification of *qhorn* queries was studied in [1]. *qhorn* is a special class of Boolean quantified queries whose underlying form is conjunctions of quantified Horn expressions. In order to verify that a given query is equivalent to the one intended by a user, a unique *verification set* of polynomial size is constructed. The verification set consists of examples uniquely determining the given query. The verification algorithm classifies some questions in the set as answers (positive examples) and others as non-answers (negative examples). The query is incorrect if the user disagrees with any of the query's classification of questions in the verification set.

A number of notions of characterizability has been studied in the context of grammatical inference [6]. Their work is related to Gold's classical model [7, 8]. In the first approach characteristic samples are constructed for a given algorithm. The algorithm must return a concept consistent with the given sample and for any sample extending the characteristic sample for the concept c, it has to return c. In another variant characterizability is parametrized with a set I of algorithms, i.e., for each concept c, its characteristic set must allow any algorithm from I to identify c. Finally, they introduce the following notion: a concept class C is *polynomially characterizable* iff for each concept c there exists a characteristic sample S of polynomial size such that if another non-equivalent concept c' is compatible with S then c is not compatible with the characteristic sample for c'. In our case q is the only query that is consistent with the characteristic sample for q.

In the learning scenario the main objective is to find a learning algorithm that produces a formula (query) consistent with a given set of examples [7, 8, 6, 19, 20]. One typically uses a weaker notion of characteristic sample w.r.t. a given learning algorithm. This allows learner bias, *collusion*, e.g., using a fixed order on the alphabet in language inference. Characterizability is stronger because it implies the existence of a characterizing sample independent of the learning algorithm (no bias).

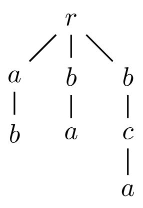
Organization. In Section 2 we recall basic notions on XML and twig queries. In Section 3 we formalize the problem of characterizing queries with examples and show that unions of twig queries are not characterizable. In Section 4 we show that twig queries are characterizable but may require exponentially many examples. In Section 5 we recall anchored twig queries and their fundamental properties. In Section 6 we present basic generalization operations, show their connection with injective embeddings, and show how to use them to characterize anchored twig queries under the injective semantics with polynomially small examples. In Section 7 we show that the generalization operations can be used to minimize anchored twig query and then show how to extend the approach used in Section 6 to construct polynomially small sets of examples characterizing anchored twig queries in the standard semantics. In Section 8 we summarize our results and outline future directions of study.

Acknowledgments. This paper is partially supported by the Polish National Science Centre grant DEC-2013/09/B/ST6/01535.

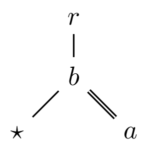
2 Basic notions

In this section we recall basic notions used to model XML documents and twig queries. Throughout this paper we assume an infinite set of node labels Σ which allows us to model documents with textual values. Also, we fix one special label $r \in \Sigma$ that we use on the root nodes of all trees and queries.

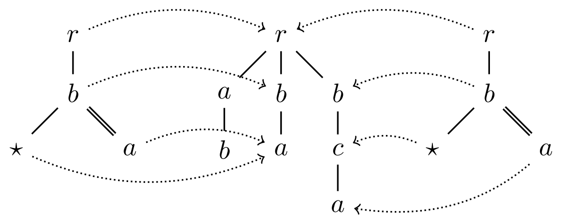
Trees. We model XML documents with unranked trees whose nodes are labeled with elements of Σ. Formally, a *tree* t is a tuple $(N_t, root_t, parent_t, lab_t)$, where N_t is a nonempty finite set of nodes, $root_t \in N_t$ is the root node, $parent_t : N_t \setminus \{root_t\} \to N_t$ is a child-to-parent function, and $lab_t : N_t \to \Sigma$ is a labeling function. By *Tree* we denote the set of all trees. An example of a tree is presented in Fig. 1a. We define a number of additional notions. A *leaf* is any node that has no children. A *path* in t from n to m (of length k) is a sequence of nodes $n = n_1, \dots, n_k = m$ such that $parent_t(n_i) = n_{i-1}$ for $1 < i \leq k$. Then we also say that n is an *ancestor* of m and m is a *descendant* of n. Note that those two terms are reflexive: every



(a) Tree t_0.



(b) Twig query q_0.



(c) Embeddings of q_0 in t_0.

Figure 1 Tree, twig query, and embeddings.

node is its own ancestor and descendant. We add the adjective *proper* to indicate that n and m are different nodes. The *depth* of a node is the length of the path from the root to the node. The *height* of a tree t, denoted $\mathit{height}(t)$, is the depth of its deepest leaf. The *size* of t, denoted $\mathit{size}(t)$, is the number of its nodes.

Queries. In general, a class of Boolean queries is a set $\mathcal{Q}$ with an implicitly given function $L : \mathcal{Q} \to 2^{\mathit{Tree}}$ that maps every query $q \in \mathcal{Q}$ to the set $L(q) \subseteq \mathit{Tree}$ of trees that satisfy q. The base class of queries, that we study in this paper, are (Boolean) twig queries, known also as *tree patterns* [2]. Basically, a twig query is an unranked tree that may additionally use a distinguished wildcard symbol $\star$ as a label and has two types of edges, child and proper descendant, corresponding to the standard XPath axes. Fig. 1b contains example of a twig queries: child edges are drawn with a single line and descendant edges with a double line.

Formally, a *twig query* q is a tuple $(N_q, \mathit{root}_q, \mathit{parent}_q, \mathit{lab}_q, \mathit{edge}_q)$, where N_q is a nonempty finite set of nodes, root_q is the *root* node, $\mathit{parent}_q : N_q \setminus \{\mathit{root}_q\} \to N_q$ is a child-to-parent function, $\mathit{lab}_p : N_q \to \Sigma \cup \{\star\}$ is a labeling function, and $\mathit{edge}_q : N_q \setminus \{\mathit{root}_q\} \to \{\mathit{child}, \mathit{desc}\}$ is the function that indicates the type of the incoming edge of a non-root node. By *Twig* we denote the set of all twig queries. We adapt the standard notions defined for trees (leaf, path, etc.) to twig queries by ignoring the edge_q component of the query.

Embeddings. We define the semantics of twig queries using the notion of an embedding which essentially maps nodes of the twig query to the nodes of the tree in a manner consistent with the semantics of the edges and the node labels. In the sequel, for two $x, y \in \Sigma \cup \{\star\}$ we say that x *matches* y if $y \neq \star$ implies $x = y$. Note that this relation is not symmetric: the label a matches $\star$ but $\star$ does not match a. Formally, an *embedding* of a twig query q in a tree t is a function $\lambda : N_q \to N_t$ such that:

1. $\lambda(\mathit{root}_q) = \mathit{root}_t$,
2. $\mathit{lab}_t(\lambda(n))$ matches $\mathit{lab}_q(n)$ for every node n of q,
3. $\lambda(n)$ is a proper descendant of $\lambda(\mathit{parent}_q(n))$ for every $n \in N_q \setminus \{\mathit{root}_q\}$,
4. $\lambda(n)$ is a child of $\lambda(\mathit{parent}_q(n))$ for every $n \in N_t \setminus \{\mathit{root}_q\}$ such that $\mathit{edge}_q(n) = \mathit{child}$.

Then, we say that t is *satisfies* q. Fig. 6 presents all embeddings of the query q_0 in tree t_0. The *language* of a query $q \in \mathit{Twig}$ is the set of all trees satisfying q

$$L(q) = \{t \in \mathit{Tree} \mid t \text{ satisfies } q\}.$$

The notion of an embedding extends in a natural fashion to a pair of queries $q, p \in \mathit{Twig}$: an *embedding* of q in p is a function $\lambda : N_q \to N_p$ that satisfies the conditions 1, 2, and 3 above (with t being replaced by p) and the following condition (which ensures that child edges are mapped to child edges only):

4.′ $\lambda(n)$ is a child of $\lambda(\mathit{parent}_q(n))$ and $\mathit{edge}_p(\lambda(n)) = \mathit{child}$ for every $n \in N_q \setminus \{\mathit{root}_q\}$ such that $\mathit{edge}_q(n) = \mathit{child}$.

Then, we write $q \preccurlyeq p$. Because a tree can be seen as a twig query, we often abuse the notation and write $t \preccurlyeq q$ to indicate that there is an embedding of q in t.

Query containment and equivalence. Given two queries q and p, q is *contained* in p, in symbols $q \subseteq p$, iff $L(q) \subseteq L(p)$. We say that q and p are *equivalent*, denoted $q \equiv p$, if $L(q) = L(p)$. It is well known that for twig queries, the existence of an embedding implies containment but the converse does not hold in general [13]. There are also significant computational differences: the containment of twig queries is coNP-complete [17, 15] whereas testing the existence of an embedding is in PTIME.

3 Characterizing queries with examples

In this section we formally define characterizability of queries and show that unions of twig queries are not characterizable.

An *example* is an element of $\mathit{Tree} \times \{+, -\}$, a pair consisting of a tree and an indicator of whether the example is *positive* $(+)$ or *negative* $(-)$. Every query q defines the set of its examples $L^{\pm}(q)$:

$$L^{\pm}(q) = L(q) \times \{+\} \cup (\mathit{Tree} \setminus L(q)) \times \{-\}.$$

Given a set of examples S, we denote by $S_+ = \{t \in \mathit{Tree} \mid (t, +) \in S\}$ and by $S_- = \{t \in \mathit{Tree} \mid (t, -) \in S\}$ the sets of respectively positive and negative examples in S. A query q is *consistent* with examples S iff $S_+ \subseteq L(q)$ and $S_- \cap L(q) = \emptyset$ (or simply $S \subseteq L^{\pm}(q)$).

▶ **Definition 3.1.** A class of queries $\mathcal{Q}$ is *characterizable* iff for every query $q \in \mathcal{Q}$, there exists a finite set of examples $\mathit{Char}(q)$ characterizing q i.e., such that q is the only query in $\mathcal{Q}$ consistent with $\mathit{Char}(q)$ (modulo query equivalence).

Characterizability alone does not ensure any bound on the cardinality of the set of characterizing examples nor any bound on their size. As we show later on, twig queries are characterizable but the number of necessary examples may be exponential, which may be undesirable for practical purposes. Therefore, we formalize a variant of characterizability that ensures a more manageable size of the characterizing set of examples.

▶ **Definition 3.2.** A class of queries $\mathcal{Q}$ is *succinctly characterizable* iff there exists a polynomial $\mathit{poly}(x)$ such that for every query $q \in \mathcal{Q}$ there exists a set of examples $\mathit{Char}(q)$ characterizing q and such that its cardinality is bounded by $\mathit{poly}(\mathit{size}(q))$ and so is the size of every of its elements.

3.1 Non-characterizability of unions of twig queries

We now consider the class *UTwig* consists of finite subsets of twig queries $Q = \{q_1, \ldots, q_k\} \subseteq \mathit{Twig}$ interpreted in the natural fashion: $L(Q) = \bigcup\{L(q) \mid q \in Q\}$. The *height* of a nonempty union of twig queries Q is the maximum height of a query in Q. Note that if the height of tree t is superior to the height of a twig query q, then q is not satisfied in t. Naturally, the same necessary condition hold for unions of twig queries.

We say that a query $Q \in \mathit{UTwig}$ is *unsaturated* if the set of its negative examples $\mathit{Tree} \setminus L(Q)$ is infinite and contains trees of arbitrary height. The universal query $\{\}$ is not unsaturated because it has no negative examples. One can, however, easily see that *UTwig* contains unsaturated queries e.g., the singleton query $Q_0 = \{q_0\}$ with q_0 from Fig. 1b is unsaturated because any tree that whose root node does not have a b child is a negative example of Q_0.

Now, take any unsaturated query $Q \in \mathit{UTwig}$ and a set of examples S consistent with Q. Let $U_- = (\mathit{Tree} \setminus L(Q)) \setminus S_-$ be the set of all negative examples of Q that are not used in S. From U_- we pick any element t whose height is greater than the height of any negative example in S. We treat t as a twig query and construct $Q' = Q \cup \{t\}$. Clearly, Q' is consistent with S because all positive examples S_+ are satisfied by $Q \subseteq Q'$ and none of the negative examples satisfy the newly added query component t because the height of t is greater than the height of any of the negative examples. Also, Q and Q' are not equivalent because t satisfies Q' but not Q.

▶ **Theorem 3.3.** *Unions of twig queries are not characterizable.*

Finally, we point out that when applied with diligence the above procedure can be iterated infinitely thus generating an infinite sequence of queries consistent with the given set of examples.

4 Characterizability of Twig queries

In this section we show that Twig queries are characterizable but may require a number of examples exponential in the size of the query.

▶ **Proposition 4.1.** *Twig queries are characterizable.*

Proof. For a tree $t \in \mathit{Tree}$ we define the *Twig*-theory it generates $\mathit{Th}(t) = \{q \in \mathit{Twig} \mid t \in L(q)\}$, the set of all twig queries that are satisfied by t. First, we show that for any tree its *Twig*-theory is finite modulo query equivalence. We observe that the height of any query $q \in \mathit{Th}(t)$ is bound by the height of t and the number of different labels in q is also bounded by the number of different labels in t. Because we consider only non-equivalent members of $\mathit{Th}(t)$, no node of q has two identical subtrees rooted at any two children of the node. The two observations above allow us to inductively prove that the number of non-equivalent different queries in $\mathit{Th}(t)$ is indeed finite.

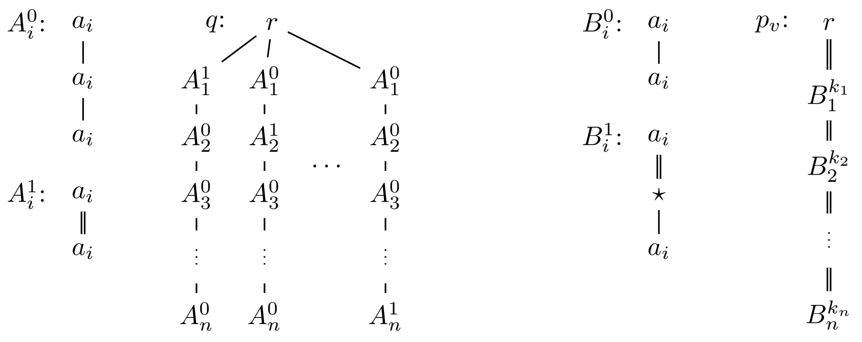
Next, for a given twig query q we outline the construction of a characterizing set of examples. For this, we construct a positive example t_0^q obtained from q by replacing every descendant edge by a child edge and using a fresh label a_0 (not used in q) to replace every $\star$. Note that $q \in \mathit{Th}(\mathsf{t}_0^q)$ and that for any $p \in \mathit{Th}(\mathsf{t}_0^q)$ that is not equivalent to q there exists a witness t of the non-equivalence of p and q, which we can use as a positive example, if t satisfies q but not p, or as a negative example, if t satisfies p but not q. Since $\mathit{Th}(\mathsf{t}_0^q)$ contains a finite number of queries modulo equivalence, only a finite number of examples is necessary to construct a set of examples characterizing q. ◀

Now, we show that twig queries may require exponential number of examples to characterize them.

▶ **Proposition 4.2.** *For any natural number n there exists a twig query q such that any set characterizing q contains at least 2^n examples.*

Proof. For a given natural number n we construct a query q and a set of twig queries U such that:

1. for each $p \in U$ we have $p \subseteq q$;


A_i^0: a_i a_i a_i
A_i^1: a_i a_i
q: r
A_1^1 A_2^0 A_3^0 A_n^0
A_1^0 A_2^1 A_3^0 A_n^0
A_1^0 A_2^0 A_3^0 A_n^1
B_i^0: a_i a_i
B_i^1: a_i $\star$ a_i
p_v: r $B_1^{k_1}$ $B_2^{k_2}$ $B_n^{k_n}$

(a) Query q **(b)** Query p_v for $v = (k_1, \ldots, k_n)$.

Figure 2 Twig query q requiring exponentially many examples w.r.t. p_v.

2. for each $p \in U$ we have $q \not\subseteq p$; Moreover, no single positive example t can witness the fact that any two distinct $p_1, p_2 \in U$ are not equivalent to q;
3. U contains 2^n queries.

Hence any set of examples S characterizing q will have at least 2^n negative examples used to distinguish q from queries in U. We construct the query q over the set of labels $\{a_1, \ldots a_n, r\}$ as illustrated in Fig. 2a.

We also construct an auxiliary set of queries W consists of patterns p_v with v ranging over all $\{0,1\}$-vectors of length n, constructed as presented in Fig. 2b. We show that the set W satisfies the conditions 2 and 3 and later we use it to construct the set U that satisfies all conditions.

Clearly, for every vector v we have $q \not\subseteq p_v$ essentially because $A_i^1 \not\subseteq B_i^k$ for any $i \in \{1, \ldots, n\}$ and $k \in \{0,1\}$. Now, take two $\{0,1\}$-vectors $v = (k_1, \ldots, k_n)$ and $w = (k'_1, \ldots, k'_n)$ that differ at a position $j \in \{1, \ldots, n\}$ and w.l.o.g. assume that $k_j = 0$ and $k'_j = 1$. Suppose now that there exists a tree t that witnesses both the facts $q \not\subseteq p_v$ and $q \not\subseteq p_w$ i.e., t satisfies q but neither p_v and p_w. Let s_j be the j-th branch of q i.e., the branch using A_j^1. Since there is an embedding of q into t, take the path in which the branch s_j is embedded into

$$\pi = r \cdot a_1 \cdot a_1 \cdot a_1 \cdot \ldots a_{j-1} \cdot a_{j-1} \cdot a_{j-1} \cdot a_j \cdot \tau \cdot a_j \cdot a_{j+1} \cdot a_{j+1} \cdot a_{j+1} \cdot \ldots a_n \cdot a_n \cdot a_n,$$

where τ is a possibly empty path fragment. Note that if τ is empty, then $\pi \preccurlyeq p_v$, and thus, $t \preccurlyeq p_v$. However, if τ is not empty then $\pi \preccurlyeq p_w$, and thus, $t \preccurlyeq p_w$. This contradicts the assumption that t does not satisfy both p_v and p_w. Consequently, every query $p \in W$ requires a unique positive example to distinguish it from q.

Now, for two twig queries p and q by $p \cap q$ we denote the twig query obtained by joining p and q at the root node (which has the same label). The set of queries U is defined as $U = \{q \cap p \mid p \in W\}$. Because $L(p \cap q) = L(p) \cap L(q)$, every element of U is properly contained in q and every element of U still requires a separate example to distinguish it from q. ◀

5 Anchored twig queries

In this section, we present the class of anchored twig queries and argue that they constitute a subclass that is functionally very close to twig queries. We also present the construction

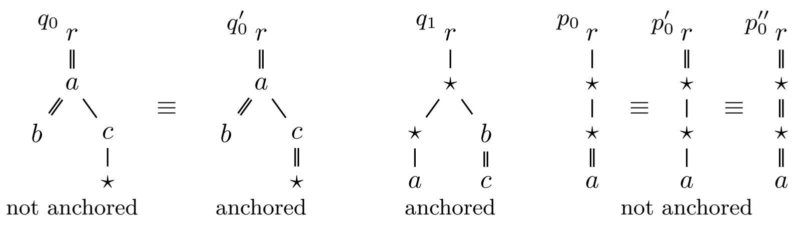


Figure 3 Anchored and non-anchored twig queries.

of representative documents for a given anchored twig query and recall the relationships between their structure and semantics with their important computational implications.

The class of anchored twig queries imposes restrictions on the mutual use of $\star$ and the descendant edges.

▶ **Definition 5.1.** A twig query is *anchored* if the following two conditions are satisfied:

1. A $//$-edge can be incident to a $\star$-node only if the node is a leaf.
2. A $\star$-node may be a leaf only if it is incident to a $//$-edge.

By *AnchTwig* we denote the set of all anchored twig queries.

A number of anchored and non-anchored queries is presented in Fig. 3. Note that the second condition requiring a $\star$ leaf to be incident to a descendant edge is merely technical: if the rule is violated by some $\star$ leaf, we can change its edge to a descendant edge and obtain an equivalent query (cf. $q_0 \equiv q'_0$ in Fig. 3).

In essence, anchored twig queries do not allow the descendant edges touch $\star$ except for leaves and thus cannot express conditions on ancestry of nodes with a minimal distance between them. For instance the query p_0 (Fig. 3) checks for the existence of a node labeled a at depth ≥ 2. We do not believe this restriction to be significant one from the practical point of view.

The reason we use anchored queries is the close relationship between the structure of the query and its semantics [19]: containment is equivalent to the existence of embedding. More precisely, for any $p, q \in \mathit{AnchTwig}$ we have $p \subseteq q$ iff $p \preccurlyeq q$. The same relationship does not hold for queries that are not anchored e.g., the non-anchored query q_0 and the anchored query q'_0 in Fig. 3 are equivalent and in particular $q'_0 \subseteq q_0$ but there is no embedding of q_0 into q'_0. Consequently, testing the containment is reduced to testing the existence of an embedding, and therefore, is in PTIME. This stands in contrast with coNP-completeness of the containment of arbitrary twigs [17, 15].

The main tool used in proving the equivalence of containment and embedding for anchored queries are *containment characterizing trees* [13, 19]. For an anchored query q of height k we construct two trees:

t_0^q is obtained from q by replacing every descendant edge by a child edge and every $\star$ with a fresh label a_0 that is not used by q.

t_1^q is obtained from q by replacing every $\star$ by a_1 and every descendant edge by a a_2-path of length k, a_1 and a_2 are two different fresh labels not used in q and different from a_0.

An example of the construction is presented in Fig. 4.

The instrumental result follows.

▶ **Lemma 5.2** ([19]). *Take any anchored query q and construct t_q^1 as described above. For any query p whose height is bounded by the height of q and that does not use the labels a_1 and a_2, $\mathsf{t}_1^q \preccurlyeq p$ implies $q \preccurlyeq p$.*

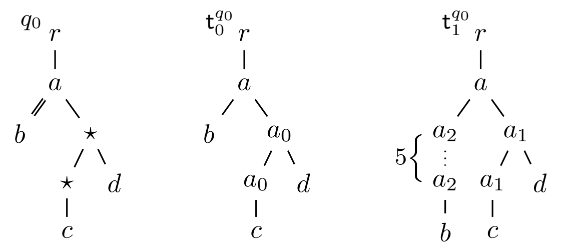


Figure 4 Construction of containment characterizing trees.

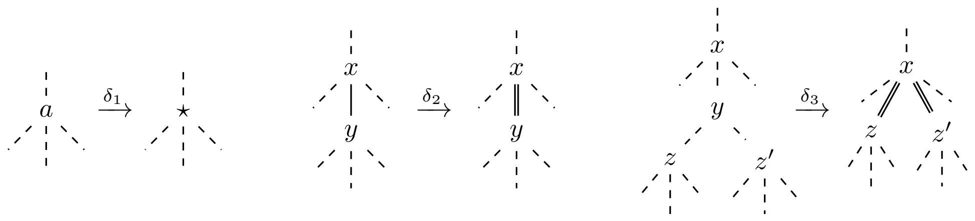


Figure 5 One-step generalization operations ($\rightarrow$).

The proof consists of an *anchoring* technique that normalizes the embedding of p into t_1^q and then translates it to an embedding of p into q. Note that if $t_0^q \preccurlyeq p$, then the height of p is bounded by the height of q and p does not use a_1 and a_2. Therefore if both t_0^q and t_1^q satisfy p, then $q \subseteq p$.

The most important implication of Lemma 5.2 is that by using only two positive examples t_0^q and t_1^q we only need to care about the queries that properly contain q and provide negative examples to distinguish q from queries properly containing q. Although their number might be quite large, we focus only on the most specific ones:

$$\Phi(q) = \{q' \in \mathit{AnchTwig} \mid q \subsetneq q' \wedge \nexists q''.\ q \subsetneq q'' \subsetneq q'\}.$$

From the view of the semi-lattice of anchored twig queries, we wish to gain an understanding of its topology to characterize the (outbound) neighborhood of a query.

6 Generalization operations

In this section, we introduce a set of generalization operations and show their connection with an injective embeddings and how the operations allow us to navigate the semi-lattice of anchored twig queries under injective semantics. We use those findings to succinctly characterize anchored twig queries under the injective semantics.

We employ simple generalization operations that we first define on arbitrary twig queries and later on tailor them to anchored twig queries. There are 3 operations, presented in Fig. 5, where dashed lines indicate optional arbitrary edges, x, y and z may have arbitrary labels in $\Sigma \cup \{\star\}$ while a ranges over Σ:

δ_1 changes the label of a non-$\star$ node to $\star$;
δ_2 changes a child edge to a descendant edges;
δ_3 removes a node and connects all its children to the parent node with a descendant edge;

We say that q is *one-step generalization* of p, in symbols $p \rightarrow q$, iff q is obtained by performing one of the 3 generalization operations.

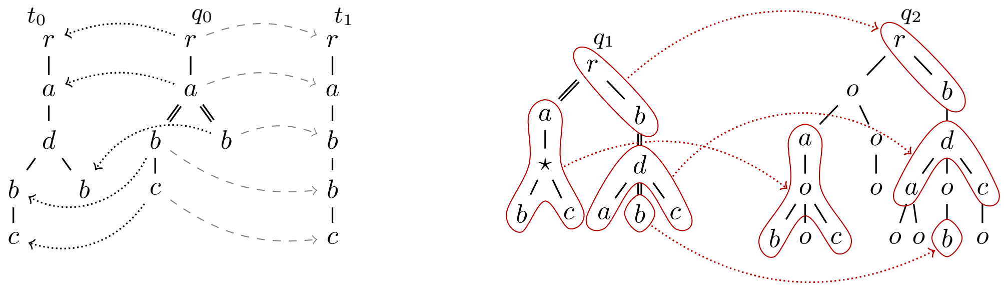


Figure 6 Injective embeddings $t_0 \preccurlyeq_\circ q_0$ and $q_2 \preccurlyeq_\circ q_1$.

6.1 Injective embeddings

There is a close connection between applying sequences of generalization operations and the existence of a special kind of injective embeddings. Formally, an embedding of λ of q into p is *injective* if it additionally satisfies the condition:

5. $\lambda(n_1)$ is an ancestor of $\lambda(n_2)$ in p if and only if n_1 is an ancestor of n_2 in q, for any two nodes n_1 and n_2 of q.

We write $p \preccurlyeq_\circ q$ if there is an injective embedding of q into p. We define analogously the injective embeddings of a twig query into a tree. Fig. 6 presents an example of an injective embedding of q_0 into t_0.

We point out that any injective embedding is an injective function but the converse does not necessarily holds (cf. q_0 and t_1 in Fig. 6). This however will not lead to confusion as we do not consider embeddings that are injective functions while violating condition 5. We define the *injective semantics* of twig queries as $L_\circ(q) = \{t \in \mathit{Tree} \mid t \preccurlyeq_\circ q\}$ and by $q \subseteq_\circ p$ denote $L_\circ(q) \subseteq L_\circ(p)$. The anchoring technique of Lemma 5.2 can be easily adapted to injective embeddings because injective embeddings are under closed composition.

▶ **Corollary 6.1.** *For any $p, q \in \mathit{AnchTwig}$, $p \subseteq_\circ q$ iff $p \preccurlyeq_\circ q$ iff $p \to^* q$.*

The connection between the generalization operations and injective embeddings is quite natural. While it is quite obvious that $p \to^* q$ implies $p \preccurlyeq_\circ q$, the converse is also true because the existence of an injective embedding of q into p ensures that all fragments of q match areas of p in a configuration that allows to easily identify the generalization operations required to transform p into q, cf. the injective embedding of q_1 into q_2 in Fig. 6. Hence,

▶ **Lemma 6.2.** *For arbitrary twig queries $p, q \in \mathit{Twig}$ we have $p \to^* q$ iff $p \preccurlyeq_\circ q$.*

Finally, we point out, however, alternative types of injective embeddings have been identified and used in the literature (cf. [12]), for our purposes any type of injective embeddings can be used. However, the set of necessary basic generalization operations (Fig. 5) depends on the chosen type of injective embedding, and we have chosen the type of injective embeddings known as *ancestor-preserving* embeddings because the generalization operations seems to be the most natural.

6.2 Generalizations of anchored twig queries

We wish to use the connection between generalization operations and the containment in order to map out the semi-lattice $\langle \mathit{AnchTwig}, \subseteq_\circ \rangle$ of anchored twig queries under the injective

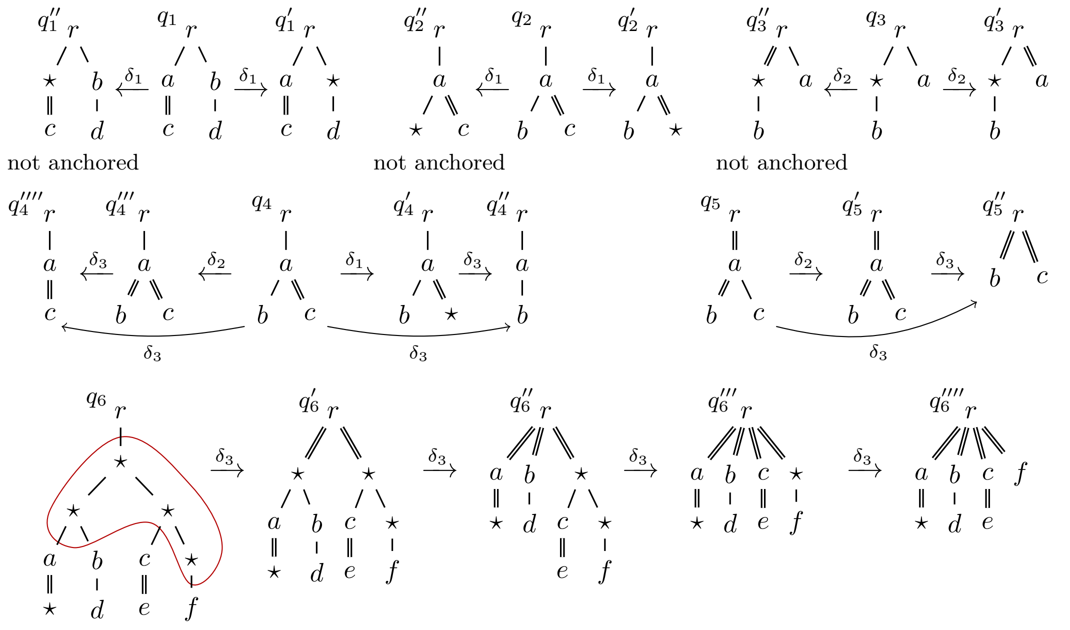


Figure 7 Applying generalization operations to anchored twig queries.

semantics. More precisely, for a given anchored query p we wish to know the set of anchored queries $\Phi_\circ(p)$ in the immediate (outgoing) neighborhood of p.

$$\Phi_\circ(q) = \{q' \in \mathit{AnchTwig} \mid q \subsetneq_\circ q' \wedge \nexists q'' \in \mathit{AnchTwig}.\ q \subsetneq_\circ q'' \subsetneq_\circ q'\}.$$

Lemma 6.2 encourages us to approach this challenge with the use of generalization operations tailored to anchored queries.

Given two anchored queries $p, q \in \mathit{AnchTwig}$, we say that q is an *immediate anchored generalization* of p, in symbols $p \lhd_\circ q$, iff $p \rightarrow^+ q$ and there is no $z \in \mathit{AnchTwig}$ such that $p \rightarrow^+ z$ and $z \rightarrow^+ q$. Essentially, the immediate anchored generalization relation defines the Hasse diagram of the semi-lattice of the anchored twig queries under injective semantics, and therefore, $\Phi_\circ(p) = \{p' \in \mathit{AnchTwig} \mid p \lhd_\circ p'\}$. We next show how $\lhd_\circ$ can be defined in terms of a small number of macros consisting of sequences of generalization operations.

To obtain an immediate anchored generalization of an anchored twig query we apply diligently the generalizations operations, making sure that: 1) the end result is an anchored twig query and 2) there is no intermediate anchored twig query that can be obtained on an alternative path. In particular, the first two generalization operations δ_1 and δ_2 can be used as long as they do not yield a query that is not anchored (cf. q_1, q_2, and q_3 in Fig. 7). The δ_3 is more involved. First of all, under certain conditions applying δ_3 is not authorized because an intermediate query can be reached with operations δ_1 or δ_2 (cf. q_4 and q_5 in Fig. 7). Furthermore, while possibly violating the structural constraints of anchored queries, δ_3 can be applied to a non-leaf $\star$ node provided that δ_3 is applied to all neighboring $\star$-nodes (cf. q_6 in Fig. 7).

We state formally the manner in which the generalizations should be used on anchored twig queries.

▶ **Lemma 6.3** ($\lhd_\circ$-operations). *For any anchored twig queries $p, q \in \mathit{AnchTwig}$ we have that $p \lhd_\circ q$ iff q is obtained from p by applying:*

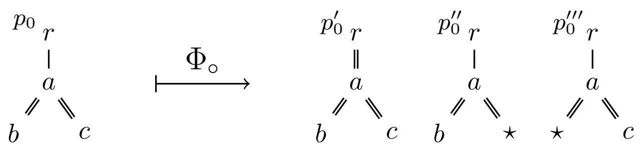


Figure 8 Immediate anchored generalizations.

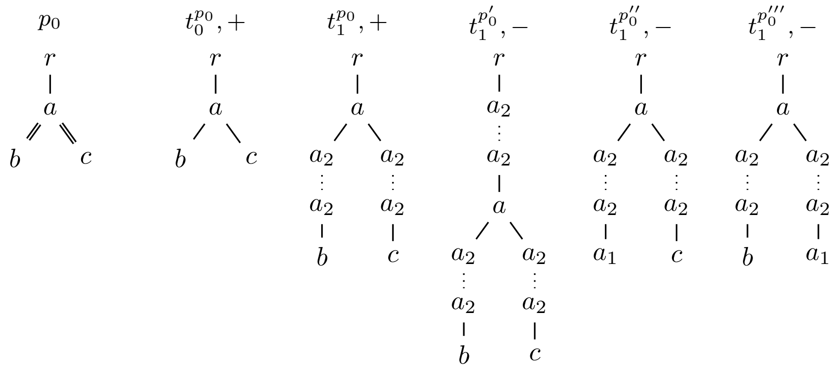


Figure 9 Characterizing query under the injective semantics.

1. δ_1 *to an inner node incident to child edges only;*
2. δ_1 *to a leaf node incident to a descendant edge;*
3. δ_2 *to an edge that is not incident to a* $\star$ *node;*
4. δ_3 *to a leaf node only if it is a* $\star$ *node or if its parent is a* $\star$ *node with other children;*
5. δ_3 *to a non-*$\star$ *node incident to descendant edges only;*
6. δ_3 *in a exhaustive manner to a connected area of* $\star$ *nodes.*

In the sequel we refer to the macros from Lemma 6.3 as $\triangleleft_\circ$-operations. We point out that from the point of view of $\triangleleft_\circ$-operations, any connected area of $\star$ nodes is seen as one particular node and we identify it with its top most $\star$ node. Note that the number of possible different applications of $\triangleleft_\circ$-operations to a query p is $O(size(p))$, and consequently, the $\Phi_\circ(p)$ contains a polynomial number of queries each of size bounded by the size of p. An example of construction of $\Phi_\circ$ is presented in Fig. 8.

6.3 Characterizability of AnchTwig under injective semantics

Because the number of possible immediate anchored generalizations of a query p is $O(size(p))$, we can characterize any anchored twig query p under injective semantics with the following set of examples (with all t_1^q's trees using a_2-chains of the same length as t_1^p):

$$Char_\circ(p) = \{(\mathsf{t}_0^p, +), (\mathsf{t}_1^p, +)\} \cup \bigcup\{(\mathsf{t}_1^q, -) \mid q \in \Phi_\circ(p)\}.$$

An example of construction of the characterizing set of examples is presented in Fig. 9 for the query p_0 from Fig. 8.

To show that $Char_\circ(p)$ does indeed characterizes p, take any query q consistent with $Char_\circ(p)$. Since q is satisfied by both t_0^p and t_1^p, $p \preccurlyeq_\circ q$ and if p would be properly contained by q, then q would satisfy one of the negative examples in $Char_\circ(p)$.

▶ **Theorem 6.4.** *Anchored twig queries are succinctly characterizable under injective semantics.*

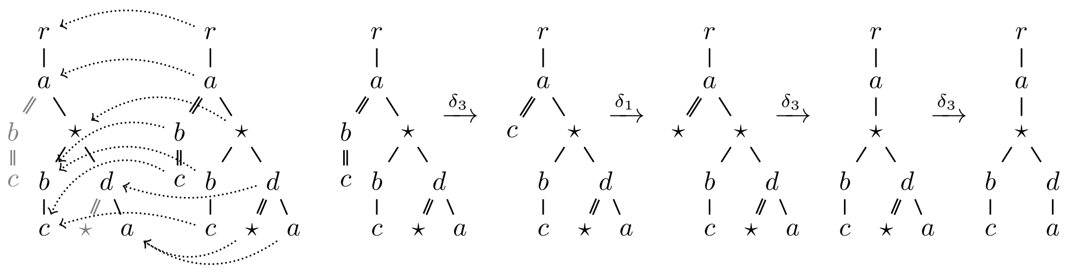


Figure 10 Reducing a query.

7 Characterizability of Anchored Twig queries

In this section, we extend the approach presented in the previous section to the standard semantics of twig queries. The main challenge is to handle possible overlaps of non-injective embeddings and we introduce a duplication operation whose controlled use ensures succinct characterizability. There is, however, a lesser challenge that we need to handle first, making sure that applying a $\lhd_\circ$-operation yields a more general query. Solving this challenge also shows that minimization of anchored twig queries is tractable and can be implemented using generalization operations which further illustrates the good behavior of the class of anchored twig queries.

7.1 Reducing anchored twig queries

If we are to base the solution of characterizability on $\lhd_\circ$-operations, we must be aware of the difference between the two semantics, how it can affect the use of $\lhd_\circ$-operations, and how to overcome the difficulties that arise.

The difference between the semantics can be illustrated on the example of query q_0 in Fig. 1b: under the injective semantics it ensures the existence of a node b with at least two different children, while such constraint cannot be expressed with the standard semantics (note that neither of the embeddings in Fig. 6 and is ancestor-preserving). In fact, the leaf $\star$ node in the query q_0 is redundant, can be removed, and a smaller equivalent query (under the standard semantics) is obtained.

The implications on use of $\lhd_\circ$-operations are as follows: if we take an anchored query p and any p' obtained by applying a $\lhd_\circ$-operation., then p needs not properly included in p' because p and p' may be equivalent under the standard semantics. To address this obstacle we *reduce* the query p: iteratively apply generalization operations (cf. Fig. 10), following $\lhd_\circ$ at each step, as long as the query remains equivalent to the original one. Note that a $\lhd_\circ$-operation may remove some nodes, rename nodes to $\star$, and change child edges to descendant edges. Essentially, it is a monotone process that finishes after $O(\mathit{size}(p))$ steps. An anchored twig query p is *reduced* if it is the end result of this procedure, or more precisely, if for no $p' \in \Phi_\circ(p)$ we have $p' \preccurlyeq p$.

Interestingly, not only are reduced queries suitable for use in $\Phi_\circ$ and can be obtained efficiently, but also they are the unique minimal canonical representatives of all equivalent queries.

Indeed, we show that an anchored twig query p is not minimal iff there exists an embedding of p into p other than the identity map, and furthermore it maps at least two nodes to the same target (an overlap). We can then carefully construct the image p' of this embedding, an anchored query whose set of nodes is the range of the embedding (see example in Fig. 10).

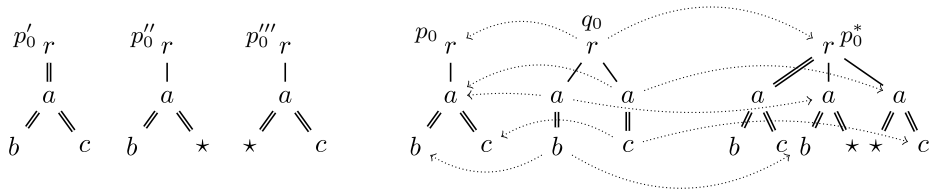


Figure 11 Immediate anchored generalizations versus an embedding with overlap $p_0 \preccurlyeq q_0$.

This query is smaller than p and the identity map is an injective embedding of p' into p, thus $p \lhd_\circ^* p'$. We point out that tractability of minimization of anchored twig queries is not a novel result. [11] presents a class of twig queries properly containing *AnchTwig* for which minimization is tractable. While the technique is similar to the presented above (but not identical), it can possibly be used as a more efficient alternative only because we shown that our reducing method also produces the minimal query.

▶ **Theorem 7.1.** *For every anchored twig query q there exists a unique reduced equivalent anchored twig query. Furthermore, this query is the size-minimal query equivalent to q and can be obtained in time polynomial in the size of q.*

7.2 Duplication operation

From now on, we use the standard semantics only. While using $\lhd_\circ$-operations on a reduced query p does construct a set of most specific queries properly containing p, it does not contain all such queries. Take for instance the query p_0 in Fig. 11 together with its immediate anchored generalizations $\Phi_\circ(p_0) = \{p_0', p_0'', p_0'''\}$. Now take the query q_0 (Fig. 11) that has an embedding into p_0, is not equivalent to p_0 (there is no embedding of p_0 into q_0), and note that it cannot be embedded into any of the immediate anchored generalizations of p_0.

This is because the embedding of q_0 into p_0 overlaps at two a nodes of q_0 whose subqueries have been obtained with applying different generalization operations.

We address the problem of the overlap with what could be seen as a adding a *duplication operation* to our system: this operation replaces a subquery rooted at a given node by a number of identical copies (including the same type of the edge to the parent). Such an operation would not, however, yield a query more general than the original query, but merely an equivalent query with redundancies, and therefore, not even reduced. Consequently, we investigate an augmented variant that applies the duplication operation and then generalizes every copy. In Fig. 11 the query p_0^* is a query obtained as a result of applying this operation to p_0 at node a. We point out, however, that the definition of this operation is not sufficiently precise: it is mutually dependent on the definition of generalization, which ideally should use the new operation. Also, unlike previously used operations which could only decrease the size of the input query, this operation has the potential to increase the size the query and when used recursively multiple time, the bounds on the size of the output query are not clear. We next settle these concerns with an appropriate recursive definition of query generalization.

We define generalizations of a query recursively on its subqueries. A subquery p of q at a node $n \in N_q \setminus \{root_q\}$ is essentially the query rooted at n but having additionally the incoming edge (from the parent) which can be altered by applying $\lhd_\circ$-operations to the root node of q. Naturally, we apply only those operations that are allowed in the context of the complete query q (thus as if knowing the label of the parent of the root node of p).

We fix a query q and let p be its subquery. A *minimal generalization* of p is any query obtained by either:

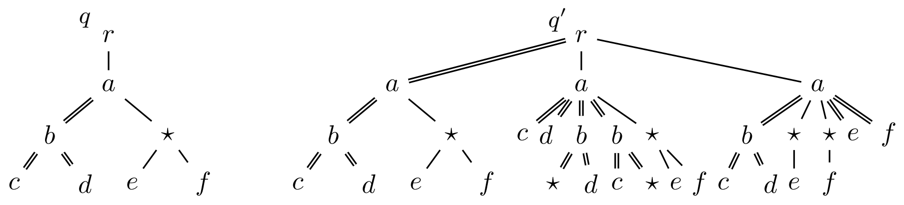


Figure 12 Constructing the minimal generalization q' of a query q ($q \lhd q'$).

1. applying a $\lhd_\circ$-operation at the root node of p (appropriately to the context of the root node in q)
2. replacing the subquery p' rooted at a child of the root node of p with the set of all *minimal generalizations* of p'.

The *minimal generalizations* of q are obtained by using only the second part of the definition (because we do not apply generalization operation to the root node). We write $q \lhd q'$ to indicate that q' is a minimal generalization of q. An example of constructing the minimal generalization of a query is presented in Fig. 12.

A reduced anchored twig query q has at most a linear number of minimal generalizations (equal to the number of children of the root node). It is not clear how big they can be given that duplication takes place. We prove a polynomial bound on the size of q' such that $q \lhd q'$. Let n be a node of q, with the path $n = n_0, n_1, \ldots, n_k = root_q$ from n to the root of q, and let ℓ_i be the number of children of n_i for $i \in \{1, \ldots, k\}$. We show with an inductive proof that q' contains $O(\ell_1 + \cdots + \ell_k)$ copies of the node n. Since $\ell_1 + \cdots + \ell_k$ is bounded by the number of nodes of q, each node is duplicated at most $size(q)$ times, and therefore, q' is of size $O(size(q)^2)$.

With an inductive proof on the structure of an arbitrary embedding between two queries that are not equivalent, we show that the minimal generalizations of a query q are the only outbound neighbors of q in the semi-lattice of anchored twig queries under the standard semantics.

▶ **Lemma 7.2.** *For any anchored twig query q, $\Phi(q) = \{q' \in AnchTwig \mid q \lhd q'\}$.*

Consequently, any anchored twig query q can be characterized with a polynomially-sized set of examples $Char(q) = \{(\mathrm{t}_0^q, +), (t_1^q, +)\} \cup \bigcup\{(\mathrm{t}_1^p, -) \mid p \in \Phi(q)\}$ and their sizes are polynomially bounded by the size of q.

▶ **Theorem 7.3.** *Anchored twig queries are succinctly characterizable.*

8 Conclusions and future work

In the present paper, we have identified and studied a novel problem of characterizing queries with examples. Our results have demonstrated that characterizability is a measure of richness of the query class, which is closely related to its expressive power. We have show that while union of twig queries are not characterizable, twigs alone are but may require exponential numbers of examples. Through the study of embeddings and generalization operations we have shown that the class of anchored twig queries is characterizable with a polynomially sized sets of examples.

Future work. We envision a number of possible future directions. We would like to extend our study of embeddings to other types of queries that employ this mechanism of defining their semantics: conjunctive relational queries and regular path queries for graphs. Naturally, the goal would to be investigate the problem of characterizability of those database models. We also intend to explore the use of the proposed constructions of characterizing sets of examples in the context of grammatical inference [6, 7, 19].

References

1 A. Abouzied, D. Angluin, Ch. Papadimitriou, J. M. Hellerstein, and A. Silberschatz. Learning and verifying quantified boolean queries by example. In *Proceedings of the 32Nd Symposium on Principles of Database Systems*, PODS '13, pages 49–60. ACM, 2013.

2 S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Tree pattern query minimization. *VLDB Journal*, 11(4):315–331, 2002.

3 M. Anthony, G. Brightwell, D. Cohen, and J. Shawe-Taylor. On exact specification by examples. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 311–318, New York, NY, USA, 1992. ACM.

4 S. Cho, S. Amer-Yahia, L. V. S. Lakshmanan, and D. Srivastava. Optimizing the secure evaluation of twig queries. In *International Conference on Very Large Data Bases (VLDB)*, pages 490–501, 2002.

5 S. Cohen and Y. Y. Weiss. Certain and possible XPath answers. In *International Conference on Database Theory (ICDT)*, 2013.

6 C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27(2):125–138, 1997.

7 E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

8 E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302 – 320, 1978.

9 S. A. Goldman and M. J. Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1):20 – 31, 1995.

10 S. A. Goldman, R. L. Rivest, and R. E. Schapire. Learning binary relations and total orders. *SIAM J. Comput.*, 22(5):1006–1034, 1993.

11 B. Kimelfeld and Y. Sagiv. Revisiting redundancy and minimization in an xpath fragment. In *EDBT 2008, 11th International Conference on Extending Database Technology*, pages 61–72, 2008.

12 J. Michaliszyn, A. Muscholl, S. Staworko, P. Wieczorek, and Z. Wu. On injective embeddings of tree patterns. *CoRR*, abs/1204.4948, 2012.

13 G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *Journal of the ACM*, 51(1):2–45, 2004.

14 F. Neven. Automata, logic, and XML. In *Workshop on Computer Science Logic (CSL)*, volume 2471 of *Lecture Notes in Computer Science*, pages 2–26. Springer, 2002.

15 F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *International Conference on Database Theory (ICDT)*, pages 315–329. Springer-Verlag, 2003.

16 S. Salzberg, A. L. Delcher, D. G. Heath, and S. Kasif. Learning with a helpful teacher. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence.*, pages 705–711, 1991.

17 T. Schwentick. XPath query containment. *SIGMOD Record*, 33(1):101–109, 2004.

18 A. Shinohara and S. Miyano. Teachability in computational learning. *New Generation Comput.*, 8(4):337–347, 1991.

19 S. Staworko and P. Wieczorek. Learning twig and path queries. In *International Conference on Database Theory (ICDT)*, March 2012.

20 B. Ten Cate, V. Dalmau, and P. Kolaitis. Learning schema mappings. In *International Conference on Database Theory (ICDT)*, March 2012.