Processing Queries on Tree-Structured Data Efficiently

Christoph Koch
Lehrstuhl für Informationssysteme
Saarland University, Saarbrücken, Germany
koch@infosys.uni-sb.de

Abstract

This is a survey of algorithms, complexity results, and general solution techniques for efficiently processing queries on tree-structured data. I focus on query languages that compute nodes or tuples of nodes – conjunctive queries, first-order queries, datalog, and XPath. I also point out a number of connections among previous results that have not been observed before. The techniques belong to five groups:

1. employing orders on the nodes of the tree for efficient labeling schemes and structural joins,
2. linear-time algorithms for evaluating Horn-SAT (the datalog technique),
3. structural decomposition techniques for queries,
4. query rewriting, and
5. holistic query processing techniques that can be explained using ideas from constraint satisfaction.

1 Introduction

There is now a fair body of research on queries on tree-structured data such as HTML, XML, and LDAP. On the theoretical side, many results on the expressive power and complexity of, as well as algorithms for, query processing on trees have been obtained. There are many applications of this work, for instance in Web data management [26], information extraction [68, 6, 31], data integration and exchange [64, 22, 4], stream processing and selective data dissemination [3, 16, 62, 38, 44, 5, 53, 40, 52], telecommunications [7], digital libraries [56], computational linguistics [11, 35], and data management in science [72, 15, 14].

In this paper, I survey earlier work on the processing of queries on trees, with the goal of isolating the essential ideas and techniques that yield efficient query evaluation. I focus on query languages that compute nodes or tuples of nodes – conjunctive queries, first-order queries, datalog, and XPath. Both complexity-theoretic and algorithmic results are presented, and techniques and results are grouped by the general idea that leads to efficient query processing. I also point out a number of new connections among previous results.

The paper is structured as follows. I start by introducing and discussing tree structures and give a motivation for node labeling schemes – the efficient processing of queries on the navigational structure of trees using structural joins (Section 2). Next I introduce the query languages studied in this paper and provide some background on their relative expressiveness (Section 3). I also show how monadic datalog can be used to process queries on trees efficiently.

In the main body of the paper, I distinguish between techniques that aim at efficiency by exploiting structural properties of query and data (Section 4), techniques based on query rewriting (Section 5), and results that employ properties that lie buried somewhat deeper in the data and that yield "holistic" query processing techniques (Section 6).

- Section 4 revolves mostly around acyclicity or more generally bounded tree- or hypertree-width of either data or queries.
- The query rewriting techniques of Section 5 exploit the treeness of the data – only because of the particularities of tree data are these rewriting techniques possible – to obtain acyclic queries. The acyclicity of the query is then used to achieve efficient evaluation. In conjunction with rewriting, I address streaming algorithms, many of which employ so-called forward queries which can be obtained by query rewriting.
- In Section 6, I discuss the so-called *X-underbar property* [45], which yields efficient query evaluation for conjunctive queries on certain tree structures by guaranteeing that globally consistent solutions of queries can be found by the (local) enforcement of arc-consistency. This approach can be traced back to work on the Constraint Satisfaction problem (CSP) in Artificial Intelligence (cf. [21]). *Holistic twig join* techniques [13, 48] have also been introduced as a technique specifically for processing tree data, and actually, their underlying idea can be taken as a special case of such arc-consistency-based constraint processing techniques.

This is the focus of the three main technical sections of this paper. In Section 7, I give a summary of the complexity results known for query languages on trees.

Note: This is not a survey of XPath complexity. The main results are stated here, but for a detailed survey of this topic see [9].

I refer to [49, 39] for surveys of the complexity theoretic background used in this paper. Three kinds of complexity of query evaluation will be considered, *data complexity* (where queries are assumed to be fixed and data variable), *query complexity* (where the query is variable and the data is assumed to be fixed), and *combined complexity* (where both data and query are considered variable) [73]. In a few upper bound results I will assume a uniform machine model in which pointers can be kept in *fixed-size* registers.

2 Tree Structures

A *relational signature* (or *schema*) is a set of relation names. A *σ-structure* is a structure (or database) of signature σ. As a convention, given a structure $\mathcal{A}$, we use $A = |\mathcal{A}|$ (the name of the structure in roman font) to denote its domain and $||\mathcal{A}||$ to denote the size of the structure in a reasonable machine representation (cf. e.g. [47, 25, 55]).

An unranked tree is a tree in which each node may have an unbounded number of children. We consider unranked ordered finite trees, which correspond to the bare tree structures of the parse trees of XML documents. Such trees can be represented by two relations *Child* and *NextSibling* over the tree domain. Let $\mathit{Child}(u, v)$ be true iff v is a child of u in the tree and let $\mathit{NextSibling}(u, v)$ be true iff, for some i, u and v are the i-th and $(i + 1)$-th children of a common parent node, respectively, counting from the left (see also Figure 1).

We also consider the transitive (R^+, for relation R) as well as the reflexive and transitive closures (R^* for relation R) of the relations *Child* and *NextSibling*. The relations Child^+, Child^*, and $\mathit{NextSibling}^+$ are also known as *Descendant*, *Descendant-or-self*, and *Following-Sibling*, respectively, in the W3C XML standards [76]. Moreover, let

$$\begin{aligned} \mathit{Following}(x, y) \quad :\Leftrightarrow \quad & \exists x_0 \exists y_0 \; \mathit{NextSibling}^+(x_0, y_0) \wedge \\ & \mathit{Child}^*(x_0, x) \wedge \mathit{Child}^*(y_0, y). \end{aligned}$$

These binary tree navigation relations are commonly called *axes* [76], a convention that we follow.

Let Σ be a node labeling alphabet. Throughout the paper, if not explicitly stated otherwise, we do not assume Σ to be fixed. We allow for tree nodes to be labeled with multiple labels. The tractability results discussed in this paper support multiple labels while the NP-hardness and expressiveness results do not make use of them. We use relations $(\mathit{Lab}_a)_{a \in \Sigma}$ to represent node labels: $\mathit{Lab}_a(v)$ is true iff node v has label a.

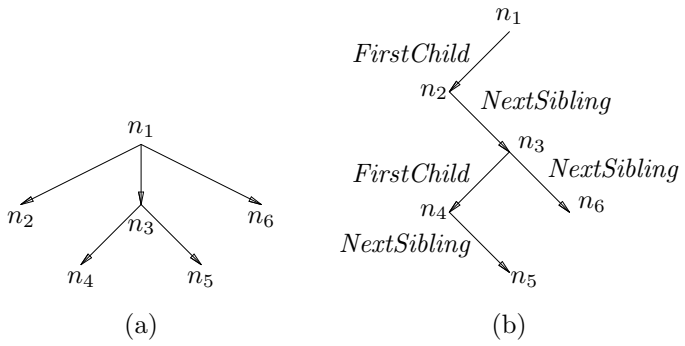


Figure 1: (a) An unranked tree and (b) its representation using the binary relations *FirstChild* ($\swarrow$) and *NextSibling* ($\searrow$).

Orders and Labeling Schemes

We consider three well-known total orders on finite ordered trees. The *pre-order* $<_{pre}$ and the *post-order* $<_{post}$ can be defined by

$$\begin{aligned} x <_{pre} y &:\Leftrightarrow Child^+(x,y) \vee Following(x,y) \\ x <_{post} y &:\Leftrightarrow Child^+(y,x) \vee Following(x,y). \end{aligned}$$

Intuitively, the pre- and postorder correspond to the order in which the opening resp. closing tag of each node of a tree is seen when reading the corresponding XML document from left to right. In XML jargon, $<_{pre}$ is also known as *document order* [76]. We also consider the ordering $<_{bflr}$ which is given by the sequence by which the nodes are visited if we traverse the tree breadth-first left-to-right.

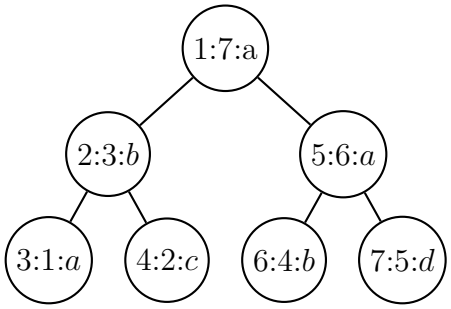
As shown above, $<_{pre}$ and $<_{post}$ can be defined from $Child^+$ and *Following*. The converse is also possible:

$$\begin{aligned} Child^+(x,y) &:\Leftrightarrow x <_{pre} y \wedge y <_{post} x \\ Following(x,y) &:\Leftrightarrow x <_{pre} y \wedge x <_{post} y \end{aligned}$$

From these axes, all others can be defined in first-order logic. Thus, a node-labeled tree can be completely represented by one triple (i,j,a) consisting of a $<_{pre}$-index i, a $<_{post}$-index j, and a label a for each node of the tree. (These indexes are chosen in a way that if two nodes u and v have, say, $<_{pre}$-indexes i and i', then $i < i'$ iff $u <_{pre} v$.)

Structural Joins

This is the starting point from which several relational storage schemes for XML data have been developed [27, 43, 42]. For instance, the extended access support relations (XASR) of [27] store a tuple consisting of the $<_{pre}$-index, the $<_{post}$-index, the $<_{pre}$-index of the parent node (NULL for the root node), and a data



(a)

R	pre	post	parent_pre	lab
	1	7	$\perp$	a
	2	3	1	b
	3	1	2	a
	4	2	2	c
	5	6	1	a
	6	4	5	b
	7	5	5	d

(b)

Figure 2: Tree (a) and XASR (b).

value or a label for each node of the data tree. Note that the $<_{pre}$-index of the parent node is redundant but allows to efficiently join a node with its parent.

Example 2.1 The tree shown in Figure 2 (a) can be represented by the XASR of Figure 2 (b). We can define the descendant axis as an SQL view

```
CREATE VIEW descendant AS
SELECT r1.pre, r2.pre FROM R r1, R r2
WHERE r1.pre < r2.pre AND r2.post < r1.post;
```

and the child axis as

```
CREATE VIEW child AS
SELECT parent_pre, pre FROM R
WHERE parent_pre is not NULL;
```

□

The advantage of such a representation is that its size is not greater than $O(||\mathcal{A}|| \cdot \log |A|)$ if A is the domain of tree $\mathcal{A}$ while computing pairs of nodes between which a descendant-relationship holds can be done by a single theta-join on the representation relation. Such joins are called *structural joins* [2]. This is clearly better than computing the transitive closure of the *Child* relation whenever such a join is to be performed (which means performing an arbitrary number of joins in a relational database management system), or storing a quadratically-sized $Child^+$ relation in the database to avoid the need to compute transitive closures.

A great number of labeling and indexing schemes for XML tree nodes have been developed, e.g. [74, 66, 63, 75], which have improved the efficiency of queries and *updates* of XML data. The observation that $<_{pre}$- and $<_{post}$-indexes are sufficient to express structural joins for all XPath axes is from [43]; however tree representations based $<_{pre}$- and $<_{post}$-indexes have been known before [23].

3 Query Languages

FO, MSO, and Conjunctive Queries

We assume familiarity with first-order logic (FO) and k-variable first-order logic FO^k (cf. [25, 55]). Monadic second-order logic (MSO) extends first-order logic by quantification over set variables, i.e., by variables ranging over sets of nodes, which coexist with first-order variables that range over single nodes. A k-ary MSO (FO) *query* is an MSO (FO) formula with k free *first-order* variables. We call the 0-ary queries *Boolean*. We use lower case node and first-order variable names and upper case names for labels, relation names, and set variables.

By a *positive* FO query, we refer to an FO query that does not employ universal quantification or negation. A *conjunctive query* is a positive FO query which in addition does not use disjunction. We will usually use the standard (datalog) *rule notation* for conjunctive queries (cf. [1]).

The *containment* of queries Q and Q' is defined in the normal way: Query Q is said to be contained in Q' (denoted $Q \subseteq Q'$) iff, for all tree structures $\mathcal{A}$, Q' returns at least all tuples on $\mathcal{A}$ that Q returns on $\mathcal{A}$. (To cover Boolean queries, tuples here may be nullary.) Two queries Q, Q' are called equivalent (denoted $Q \equiv Q'$) iff $Q \subseteq Q'$ and $Q' \subseteq Q$.

Core XPath

Many results on XPath apply to the fragment that deals only with the navigational structure of an XML document. This fragment, denoted Core XPath, consists of expressions whose input is a node and whose output is either a set of nodes or a Boolean. The latter are also referred to as *qualifiers*. We will use $p, p', \ldots$ to vary over general XPath expressions, while $q, q', \ldots$ will be used to denote qualifiers. Expressions are built up from the grammar

$$\begin{array}{rcl} p & ::= & \mathit{step} \mid p/p \mid p \cup p \\ \mathit{step} & ::= & \mathit{axis} \mid \mathit{step}[q] \\ \mathit{axis} & ::= & \mathit{arel} \mid \mathit{arel}^{-1} \mid \mathit{Self} \\ \mathit{arel} & ::= & \mathit{Child} \mid \mathit{Descendant} \mid \mathit{Descendant\text{-}or\text{-}self} \\ & \mid & \mathit{Following\text{-}Sibling} \mid \mathit{Following} \\ q & ::= & p \mid \mathrm{lab}() = L \mid q \wedge q \mid q \vee q \mid \neg q, \end{array}$$

where *axis* stands for the names of the axis relations, L denotes the labels in Σ, and $\wedge, \vee, \neg$ stand for *and* (conjunction), *or* (disjunction) and *not* (negation), respectively.

An expression p in Core XPath over a tree structure $\mathcal{D}$ is interpreted as a function $[\![p]\!]_{NodeSet}$ from a node to a set of nodes, while a qualifier q is interpreted as a unary predicate $[\![q]\!]_{Boolean} : \mathit{Nodes} \to \mathit{NodeSet}$. A unary Core XPath query

is of the form $[\![p]\!]_{NodeSet}(root)$ and thus defines a set of nodes. The semantic functions are defined inductively on the structure of p, q. For *NodeSet* expressions p we have

(P1) $[\![Self]\!]_{NodeSet}(n) := \{n\}$.
$[\![R]\!]_{NodeSet}(n) := \{n' : R(n, n')\}$.
$[\![R^{-1}]\!]_{NodeSet}(n) := \{n' : R(n', n)\}$.

(P2) $[\![step[q]]\!]_{NodeSet}(n) := \{n' : n' \in [\![step]\!]_{NodeSet}(n) \wedge [\![q]\!]_{Boolean}(n') = \text{true}\}$.

(P3) $[\![p_1/p_2]\!]_{NodeSet}(n) := \{v : \exists w \in [\![p_1]\!]_{NodeSet}(n) \wedge v \in [\![p_2]\!]_{NodeSet}(w)\}$.

(P4) $[\![p_1 \cup p_2]\!]_{NodeSet}(n) := [\![p_1]\!]_{NodeSet}(n) \cup [\![p_2]\!]_{NodeSet}(n)$.

For qualifiers q we have

(Q1) $[\![\text{lab}() = L]\!]_{Boolean}(n) :\Leftrightarrow Lab_L(n)$

(Q2) $[\![p]\!]_{Boolean}(n) :\Leftrightarrow [\![p]\!]_{NodeSet}(n) \neq \emptyset$

(Q3) $[\![q_1 \wedge q_2]\!]_{Boolean}(n) :\Leftrightarrow [\![q_1]\!]_{Boolean}(n) \wedge [\![q_2]\!]_{Boolean}(n)$

(Q4) $[\![q_1 \vee q_2]\!]_{Boolean}(n) :\Leftrightarrow [\![q_1]\!]_{Boolean}(n) \vee [\![q_2]\!]_{Boolean}(n)$

(Q5) $[\![\neg q]\!]_{Boolean}(n) :\Leftrightarrow \neg [\![q]\!]_{Boolean}(n)$

By *positive* Core XPath, we will refer to Core XPath without negation. We say that a Core XPath query is *conjunctive* if it does not use disjunction, union, or negation.

Monadic Datalog

We assume the function-free logic programming syntax and semantics of the *datalog* language known and refer to [1] for a detailed survey of datalog. *Monadic datalog* [18, 31] is obtained from full datalog by requiring all intensional predicates to be unary. For monadic datalog, one obtains a unary query by distinguishing one intensional predicate as the *query predicate*.

We use tree structures of signature

$$\tau^+ = \langle Dom,\ Root,\ Leaf,\ (\text{Lab}_a)_{a\in\Sigma}, FirstChild,\ NextSibling,\ LastSibling\rangle$$

where *Dom* is the set of nodes in the tree and the remaining predicates can be defined from *Child* and *NextSibling* in FO as follows:

$$\begin{aligned}
Root(x) &:\Leftrightarrow \neg\exists x_0\ Child(x_0, x)\\
Leaf(x) &:\Leftrightarrow \neg\exists y\ Child(x, y)\\
FirstSibling(x) &:\Leftrightarrow \neg\exists x_0\ NextSibling(x_0, x)\\
LastSibling(x) &:\Leftrightarrow \neg\exists y\ NextSibling(x, y)\\
FirstChild(x, y) &:\Leftrightarrow Child(x, y) \wedge FirstSibling(y)
\end{aligned}$$

Of course these definitions are to be considered relativized to the tree domain. (We make *Root*, *Leaf*, and *LastSibling* part of the input because datalog cannot express negation, which we used to define these relations above.)

```
algorithm Minoux(propositional Horn formula Φ)
// let Φ = (p_{1,1} ∨ ¬p_{1,2} ∨ ··· ∨ ¬p_{1,k_1}) ∧ ... ∧
//         (p_{l,1} ∨ ¬p_{l,2} ∨ ··· ∨ ¬p_{l,k_l}).
begin
  list<rule_id> rules[pred_id];
  int size[rule_id];
  pred_id head[rule_id];
  queue<pred_id> q;

  // initialization of data structures
  for 1 ≤ i ≤ l do begin
    head[i] := p_{i,1};
    size[i] := k_i − 1;
    for 2 ≤ j ≤ k_i do append i to rules[p_{i,j}];
    if k_i = 1 then append i to q;
  end;

  // main loop
  while queue is not empty do begin
    p := pop first element off q;
    output "p is true";
    for each i in rules[p] do begin
      size[i] := size[i] - 1;
      if size[i] = 0 then append head[i] to q;
    end;
  end;
end.
```

Figure 3: Linear-time algorithm for solving propositional Horn-SAT (Minoux' Algorithm).

Example 3.1 The monadic datalog program over τ^+

$$
\begin{aligned}
P_0(x) &\leftarrow \text{Label}_L(x). && (1)\\
P_0(x_0) &\leftarrow \text{NextSibling}(x_0, x),\ P_0(x). && (2)\\
P(x_0) &\leftarrow \text{FirstChild}(x_0, x),\ P_0(x). && (3)\\
P_0(x) &\leftarrow P(x). && (4)
\end{aligned}
$$

with query predicate P computes those nodes that have an ancestor labeled L. □

While monadic datalog over arbitrary finite structures is NP-complete w.r.t. combined complexity (see e.g. [31]),

Theorem 3.2 ([31]) *Monadic datalog over τ^+ has time $O(|\mathcal{P}|*|Dom|)$ combined complexity (where $|\mathcal{P}|$ is the size of the program and $|Dom|$ the size of the tree).*

This follows from the fact that all binary relations in τ^+ have bidirectional functional dependencies; for instance, each node has at most one first child and is the first child of at most one other node. Given a program $\mathcal{P}$, an equivalent ground (i.e., propositional) program can be computed in time $O(|\mathcal{P}| * |\text{Dom}|)$. Ground programs can be evaluated in linear time using Minoux' Algorithm [59], shown in Figure 3.

Example 3.3 The following Horn-SAT program is a ground version of the program of Example 3.1.

$r_1 : \text{Label}_L(3) \leftarrow;$ $\quad r_2 : \text{FirstChild}(1,2) \leftarrow;$ $\quad r_3 : \text{NextSibling}(2,3) \leftarrow;$

$d_1 : P_0(1) \leftarrow \text{Label}_L(1);$ $\quad d_2 : P_0(2) \leftarrow \text{Label}_L(2);$ $\quad r_4 : P_0(3) \leftarrow \text{Label}_L(3);$

$d_3 : P_0(1) \leftarrow P(1);$ $\quad d_4 : P_0(2) \leftarrow P(2);$ $\quad d_5 : P_0(3) \leftarrow P(3);$

$r_5 : P_0(2) \leftarrow \text{NextSibling}(2,3),\ P_0(3);$ $\quad r_6 : P(1) \leftarrow \text{FirstChild}(1,2),\ P_0(2)$

For simplicity, let us drop the rules $d_1, \ldots, d_5$ and relabel the ground atoms using integers.

$r_1 :\ 1 \leftarrow$ $\quad r_2 :\ 2 \leftarrow$ $\quad r_3 :\ 3 \leftarrow$

$r_4 :\ 4 \leftarrow 1$ $\quad r_5 :\ 5 \leftarrow 3,\ 4$ $\quad r_6 :\ 6 \leftarrow 2,\ 5$

The initialization phase of Minoux' algorithm produces the data structures

size	
r_1	0
r_2	0
r_3	0
r_4	1
r_5	2
r_6	2

head	
r_1	1
r_2	2
r_3	3
r_4	4
r_5	5
r_6	6

rules	
1	$[r_4]$
2	$[r_6]$
3	$[r_5]$
4	$[r_5]$
5	$[r_6]$
6	[ ]

q = [1, 2, 3]

The main loop of Minoux' algorithm in the first iteration takes 1 off the queue, outputs it, for each rule r in rules[1] (that is, only r_4) decrements size[r] by one, and, since size[r_4] becomes 0, appends head[r_4] = 4 to the queue. Next, it pops 2 off the queue and proceeds analogously. □

It is folklore that each monadic datalog query over arbitrary finite structures can be defined by an equivalent unary MSO query. For trees, but not for general structures, the reverse holds as well: A unary query is definable in monadic datalog over τ^+ iff it is definable in MSO over τ^+ [31].

It is known from [41] that unless the parametric-complexity equivalent of $P = NP$ holds, MSO is nonelementarily more succinct than monadic datalog on trees.[1] This is not surprising because MSO (even FO) on trees is known to be PSPACE-complete and thus considerably harder than monadic datalog on τ^+-structures.

Each monadic datalog program over trees can be efficiently rewritten into an equivalent program using only very restricted syntax.

Definition 3.4 A monadic datalog program $\mathcal{P}$ over τ^+ is in *Tree-Marking Normal Form* (TMNF) if each rule of $\mathcal{P}$ is of one of the following three forms:

$$(1)\ \ p(x) \leftarrow p_0(x). \qquad (2)\ \ p(x) \leftarrow p_0(x_0),\ B(x_0, x).$$

$$(3)\ \ p(x) \leftarrow p_0(x),\ p_1(x).$$

where the unary predicates p_0 and p_1 are either intensional or of τ^+ and B is either R or R^{-1}, where R is a binary predicate from τ^+. □

In [31], it was shown that for each monadic datalog program $\mathcal{P}$ over $\tau^+ \cup \{\mathit{Child}\}$, there is an equivalent TMNF program over τ^+ which can be computed in time $O(|\mathcal{P}|)$. In [29, 51] an automata-based technique was developed for evaluating TMNF programs in time $O(f(|Q|) + ||\mathcal{A}||)$.

Each Core XPath query can be translated into an equivalent TMNF query in linear time [29]. The slightly curious fact here is that this remains true in the presence of negation in Core XPath, for which no analogous language feature exists in datalog.

4 Exploiting Bounded Tree-Width

We recall the well-studied graph-theoretical notion of *tree-width*. Let $G = (V^G, E^G)$ be an undirected graph (edge relation E^G is symmetric). A *tree decomposition* of G is a pair (T, χ) such that T is a rooted tree with nodes V^T, χ is a function $\chi : V^T \to 2^{V^G}$ that maps each node of tree T to a subset of V^G, for each edge $(u, v) \in E^G$ there exists a node $w \in V^T$ such that $u, v \in \chi(w)$, and for each node $u \in V^G$, the set $\{v \in V^T \mid u \in \chi(v)\}$ induces a connected subtree of T. The *width* of tree decomposition (T, χ) is defined as $\big(\max\{|\chi(v)| \mid v \in V^T\}\big) - 1$. The *tree-width* of a graph G is the smallest width over all tree decompositions of G. Intuitively, graphs of low tree-width are very tree-like. As a special case, the connected graphs of tree-width one are precisely the trees. An example of a graph and a tree decomposition (of width 2) for it is given in Figures 4 (a) and (b), respectively.

[1] That is, there are MSO queries Q on trees for which the sizes of the smallest equivalent monadic datalog queries cannot be bounded by any fixed tower of exponentials $2^{2^{2^{\cdot^{\cdot^{\cdot^{|Q|}}}}}}$.

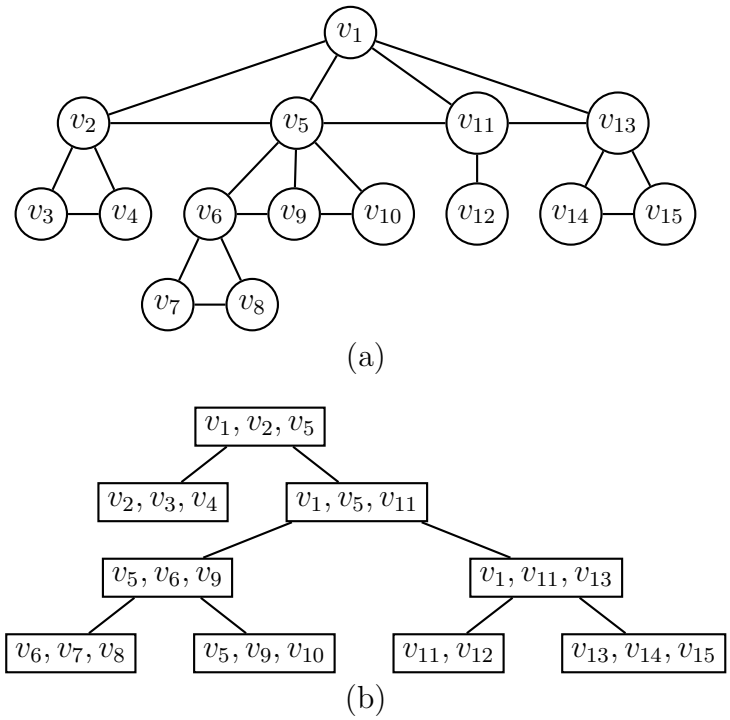


Figure 4: A (*Child*, *NextSibling*)-tree is a graph of tree-width two.

Queries

The *tree-width* of a conjunctive query Q over a vocabulary of at most binary relation symbols is defined as the tree-width of the graph $G = (V, E)$ where V consists of the variables of Q and $(x, y), (y, x) \in E$ if there is an atom $R(x, y)$ in Q. The conjunctive queries of bounded tree-width can be evaluated in polynomial time.

Theorem 4.1 ([17]) *A Boolean conjunctive query Q of tree-width k can be evaluated on a database $\mathcal{A}$ with domain A in time $O((|A|^{k+1} + ||\mathcal{A}||) * |Q|)$.*

It is known [54] that conjunctive FO^{k+1} queries have *tree-width* $\leq k$. Both this result and Theorem 4.1 generalize from conjunctive to FO queries [28]. Thus, while FO over trees is known to be PSPACE-complete with respect to combined complexity, FO^k (even over arbitrary relational structures) is known to be in time $O(||\mathcal{A}||^k * |Q|)$. Since Core XPath queries can be translated efficiently, in linear time, into equivalent FO^2 queries [57, 9], Boolean Core XPath is in time $O(||\mathcal{A}||^2 \cdot |Q|)$.

These combined complexity bounds can be improved upon by moving from the notion of tree-width to that of hypertree-width [37]. The conjunctive queries of hypertree-width 1 coincide with the so-called *acyclic* conjunctive queries (cf. e.g.

[1]). As shown by Yannakakis in [77], the acyclic (Boolean or unary) conjunctive queries can be evaluated in time $O(||\mathcal{A}|| \cdot |Q|)$. The idea of this algorithm is to process the join tree of the query bottom-up and project, as soon as possible, after each join, all the columns of the intermediate result which are not needed in subsequent joins away. The acyclicity of the query ensures that the intermediate results are always (projected) subsets of a single input relation and therefore not larger than the input. For unary queries, the join tree of the query has to be oriented in such a way that the output is a subset of a column of the input relation at the root of the join tree.

Conjunctive Core XPath queries are acyclic (see [32]) and can be evaluated using Yannakakis' algorithm both in linear time in the data *and* efficiently in the size of the query.

Proposition 4.2 ([33]) *The unary conjunctive Core XPath queries Q (with all axes) can be evaluated in time $O(||\mathcal{A}|| \cdot |Q|)$ on structures $\mathcal{A}$.*

Full XPath 1.0 was shown to be in polynomial time for combined complexity [33], and while XPath queries in general do not strictly fit into the framework of first-order queries of bounded hypertree-width, it is argued in [10, 9] that there is no second main idea for their tractability.

One may ask whether XPath is also PTIME-hard, or alternatively, whether it is in the complexity class NC and thus effectively parallelizable. Apart from theoretical interest, a precise characterization of the XPath evaluation problem in terms of parallel complexity classes may hint at what computational resources are necessarily required for query evaluation. For example, it is strongly conjectured that all algorithms for solving PTIME-hard problems actually require a polynomial amount of working memory. However, performing XPath query evaluation with limited memory resources is important in practice, e.g. in the context of data stream processing.

It was shown in [34] that Core XPath is PTIME-hard (and the proof of [34] can be easily adapted to show the PTIME-hardness of FO^2 on trees), but that already positive Core XPath is in LOGCFL w.r.t. combined complexity. For conjunctive Core XPath, this actually already follows from acyclicity and the following result:

Theorem 4.3 ([36]) *The conjunctive queries of bounded hypertree-width over arbitrary relational structures are in* LOGCFL *w.r.t. combined complexity.*

Tree Data

We say that a structure which consists only of unary and binary relations has tree-width k if the union of (the symmetric closure of) its binary relations has tree-width k.

It is well known that Boolean MSO queries on trees correspond to tree automata and have linear-time data complexity [71, 24]. A more general version of this fact for bounded tree-width structures is known as Courcelle's Theorem [19], which can be further generalized to

Theorem 4.4 ([28]) *Let* $\mathbf{C}$ *be a class of structures of bounded tree-width. For a fixed MSO formula* ϕ*, there is an algorithm that evaluates* ϕ *on each structure* $\mathcal{A} \in \mathbf{C}$ *in time* $O(||\mathcal{A}|| + ||\phi(\mathcal{A})||)$.

That is, this algorithm runs in time linear in the size of the input and the output, and in particular in linear time *in the size of the input* on MSO formulas with at most one free variable.

One can verify that unranked ordered trees represented by (*Child*, *NextSibling*)-structures have tree-width two because the union of their binary relations *Child* and *NextSibling* has tree-width two (see Figure 4, where each node v is labeled with $\chi(v)$). Transitive axis relations such as $Child^+$ or *NextSibling* (cf. Section 2) do not have bounded tree-width in general, but it is not difficult to map Core XPath queries with transitive axes to MSO over a signature with *Child* and *NextSibling* [30]. For instance, $R^*(x, y)$, where R^* is the reflexive and transitive closure of relation R, can be defined in MSO as $\forall S \left(S(x) \wedge \forall u \forall v\ S(u) \wedge R(u, v) \rightarrow S(v)\right) \rightarrow S(y)$. From this we can conclude that unary Core XPath, just like FO and MSO, is in linear time w.r.t. data complexity.

Reductions from MSO to automata do not yield good upper bounds on the combined complexity of queries, however. Indeed, they are necessarily nonelementary [58, 67]. For Core XPath, a doubly exponential translation to automata is implicit in [29, 51].

5 Query Rewriting

Each positive query on trees can also be formulated as an acyclic positive query. This is in contrast to full FO (i.e., with negation) on trees, which is known to be stronger than acyclic FO on trees resp. Core XPath [57].

Theorem 5.1 ([62, 8, 35]) *For every conjunctive query over trees there is an equivalent acyclic positive query. This query can be computed in exponential time.*

Proof. For notational simplicity, we will assume that the input query $\exists x_1 \cdots x_k\ Q$ ($k \geq 0$), with Q a conjunction of atomic formulas that uses variables $x_1, \ldots, x_k$, is Boolean. The proof, however, immediately generalizes to conjunctive queries of arbitrary arity. W.l.o.g., we assume that Q contains no *Following*-atoms. (These can be rewritten using $Child^*$ and $NextSibling^+$ atoms as described in Section 2.)

$R \setminus S$	*Child*	*Child*$^+$	*NextSibling*	*NextSibling*$^+$
Child	unsat	unsat	sat	sat
Child$^+$	sat	sat	sat	sat
NextSibling	unsat	unsat	unsat	unsat
NextSibling$^+$	unsat	unsat	sat	sat

Table 1: Satisfiability of $R(x,z) \wedge S(y,z) \wedge x <_{pre} y$ for pairs of axes R, S.

Consider the conjunctive normal form formula

$$\phi :\Leftrightarrow \bigwedge_{1 \leq i < j \leq k} (x_i = x_j \vee x_i <_{pre} x_j \vee x_j <_{pre} x_i).$$

Let Ψ be the set consisting of the $3^{\binom{k}{2}}$ disjuncts of the disjunctive normal form of ϕ. For $\psi \in \Psi$ let Q_ψ be the conjunction of atomic formulas obtained from $Q \wedge \psi$ by the following steps, in the indicated order.

1. We remove all occurrences of equality atoms $x = y$ in arbitrary order and replace, for each such atom, all occurrences of y by x.

2. For $R \in \{\mathit{Child}, \mathit{NextSibling}\}$, we remove all atoms $R^*(x,x)$ from Q_ψ and replace all occurrences of $R^*(x,y)$ (where x and y are different variables) by $R^+(x,y)$. The latter is an equivalent rewriting since Q_ψ contains either atom $x <_{pre} y$ or $y <_{pre} x$, thus x and y must map to different nodes.

3. For $R \in \{\mathit{Child}, \mathit{NextSibling}\}$, if Q_ψ contains atoms $R(x,y)$, $R^+(x,y)$ then $R^+(x,y)$ is removed from Q_ψ.

Observe that the binary atoms or Q_ψ use only *Child*, *Child*$^+$, *NextSibling*, *NextSibling*$^+$, and $<_{pre}$ as predicates. We can verify that $\exists \vec{x}\ Q_\psi$ is true if and only if $\exists \vec{x}\ Q \wedge \psi$.

Let $\mathbf{Q} = \{\exists \vec{x}\ Q_\psi \mid \psi \in \Psi\}$. Then

$$Q \ \equiv\ \exists \vec{x}\ Q \wedge \phi \ \equiv\ \bigvee \{\exists \vec{x}\ Q \wedge \psi \mid \psi \in \Psi\} \ \equiv\ \bigvee \mathbf{Q}.$$

In the following, we will call the binary relation E with

$$xEy \ :\Leftrightarrow\ \text{there is an atomic formula } R(x,y) \text{ in } Q_\psi$$

(with R a binary predicate – either an axis or $<_{pre}$) the *graph of* Q_ψ. Note that E is either cyclic or defines a total order on the variables in Q_ψ because there is an edge between any two variables of Q_ψ.

Now, for each Q_ψ of $\mathbf{Q}$, we repeat the following steps until we terminate:

- If the graph of Q_ψ is cyclic, Q_ψ is unsatisfiable and is removed from $\mathbf{Q}$. Termination. Otherwise, the graph of Q_ψ is acyclic and thus constitutes a total order of the variables in Q_ψ.

- If Q_ψ contains atoms $R(x,y), S(x,y)$ where $R \in \{\mathit{Child},\ \mathit{Child}^+\}$ and $S \in \{\mathit{NextSibling}, \mathit{NextSibling}^+\}$, Q_ψ is unsatisfiable and is removed from $\mathbf{Q}$. Termination.

- If there are no two atoms $R(x,z), S(y,z)$ in Q_ψ with x and y distinct variables and R, S different from $<_{\mathit{pre}}$ then Q_ψ is acyclic. Termination.

- We choose the pairs of atoms $R(x,z), S(y,z)$ (x and y distinct variables and R, S different from $<_{\mathit{pre}}$) such that z is maximal w.r.t. the total order given by the graph of Q_ψ. From among these, we choose a pair such that x is minimal w.r.t. the total order. By our choice, $x <_{\mathit{pre}} y$ is in Q_ψ. If $R(x,z) \wedge S(y,z) \wedge x <_{\mathit{pre}} y$ is unsatisfiable (the unsatisfiable cases can be found in Table 1), remove Q_ψ from $\mathbf{Q}$ and terminate. Otherwise, replace atom $R(x,z)$ by $R(x,y)$.

The above algorithm terminates because there are no more than $\binom{k}{2}$ non-$<_{\mathit{pre}}$-atoms and whenever we replace an atom $R(x,z)$ by an atom $R(x,y)$, y is smaller than z w.r.t. the total order. Once we have processed a pair of atoms $R(x,z)$, $S(y,z)$, we never have to process pairs of atoms $R'(x,z)$, $S'(y',z)$ for the same x and z again. Thus processing a single Q_ψ takes polynomial time and the complete rewriting of Q takes exponential time.

It can be verified that replacing $R(x,z)$ in the satisfiable cases of $R(x,z) \wedge S(y,z) \wedge x <_{\mathit{pre}} y$ by $R(x,y)$ is an equivalent rewriting:

- $R = \mathit{Child}^+$, $S \in \{\mathit{Child}, \mathit{Child}^+\}$: if x and y are ancestors of z, then $x <_{\mathit{pre}} y$ implies that x is an ancestor of y.

- $R = \mathit{NextSibling}^+$, $S \in \{\mathit{NextSibling},\ \mathit{NextSibling}^+\}$: analogous.

- $R \in \{\mathit{Child}, \mathit{Child}^+\}$, $S \in \{\mathit{NextSibling},\ \mathit{NextSibling}^+\}$: Since x is a parent/ancestor of z and y is a left sibling of z, x is also a parent/ancestor of y.

Each conjunctive query Q_ψ in the set $\mathbf{Q}$ obtained as described above is acyclic if all the $<_{\mathit{pre}}$-atoms are removed. Doing just that is an equivalent rewriting: Let Q'_ψ be the conjunction of atoms of Q_ψ excluding the $<_{\mathit{pre}}$-atoms of Q_ψ. Then $\exists \vec{x}\, Q_\psi \subseteq \exists \vec{x}\, Q'_\psi \subseteq \exists \vec{x}\, Q$; thus, $\exists \vec{x}\, Q \equiv \bigvee \mathbf{Q} \subseteq \bigvee \{\exists \vec{x}\, Q'_\psi \mid Q_\psi \in \mathbf{Q}\} \subseteq \exists \vec{x}\, Q$. $\square$

It follows that if we take the size of the query as fixed, the positive Boolean queries can be evaluated in linear time using Yannakakis' algorithm.

Corollary 5.2 *A fixed positive Boolean FO query can be evaluated on trees $\mathcal{A}$ in time $O(||\mathcal{A}||)$.*

Note that this also follows from known results on the translation of FO – or more generally, MSO – queries on trees into tree automata and the evaluation of such automata on trees. These translations work for related reasons.

Discussion and Related Work

The rewrite algorithm of the proof of Theorem 5.1 can be easily improved in two ways.

- First, in the proof above, Q was rewritten using formulas ψ that amount to all embeddings of the variables of Q into total orders of no more than k elements, plus many inconsistent formulas ϕ. Instead, [35] only makes choices of order among two variables x, y if there is a pair of atoms $R(x, z), S(y, z)$ that has to be rewritten.

- Second, using more fine-grained rewrite rules one can further reduce the number of copies of Q that have to be rewritten. The rewrite rules of [35] do not require us for each atom $\mathit{Child}^*(x, y)$ or $\mathit{NextSibling}^*(x, y)$ to first choose whether $x <_{pre} y$ to either eliminate the atom or replace it by $\mathit{Child}^+(x, y)$ or $\mathit{NextSibling}^+(x, y)$.

The special case of the above theorem that conjunctive queries using the *Child* and Child^+ axes can be rewritten into APQs is stated in [8]. Similar techniques to those of the previous proof were used in [62] to eliminate backward axes from XPath expressions and in [69] to rewrite first-order queries over trees given by certain regular path relations.

There are signatures with axes for which all conjunctive queries can be rewritten into APQs in polynomial time. It is implicit in [31] that any query in $CQ[\{\mathit{Child}, \mathit{NextSibling}\}]$ can be rewritten into an equivalent acyclic query in $CQ[\{\mathit{Child}, \mathit{NextSibling}\}]$ query in linear time. The proof of linear-time translatability of monadic datalog into TMNF uses this result: acyclic monadic datalog rules can be easily split up into rules with no more than two body atoms.

However, the translation from conjunctive queries into APQs in general is necessarily exponential: As shown in [35], there are conjunctive queries over trees, using just the Child^+ (or Child^*) axis, that cannot be polynomially translated into equivalent APQs.

Evaluating Positive Queries using XPath

By a *forward XPath* query we refer to an XPath query which uses only the forward axes *Child*, Child^+, *NextSibling*, and $\mathit{NextSibling}^+$, but none of their inverses such as *Parent* or *Ancestor*. Since the rewrite technique of the proof of Theorem 5.1 produces sets of acyclic conjunctive queries that are forest-shaped in a strong sense – there are no pairs of atoms $R(x, z), S(y, z)$ – each acyclic positive query can be rewritten into an equivalent forward Core XPath query [62].

A *streaming algorithm* scans its input data only once from left to right. The first work to present a streaming algorithm for evaluating (forward) XPath that takes only memory linear in the depth of the tree and runs in time polynomial in the size of (the data and) the query was [61]. There, the exponential size of automata is avoided by not compiling automata for managing and recognizing the subexpressions of an XPath query into a single automaton but keeping them apart, as a *transducer network*. A similar transducer-network based approach to streaming XPath processing was developed in [65]. A different algorithm for polynomial-time streaming XPath processing was presented in [50].

6 Global vs. Arc-Consistency

Let Q be a conjunctive query and let A denote the finite domain, i.e. in case of a tree the set of nodes. A pre-valuation for Q is a total function $\Theta : \mathit{Var}(Q) \to 2^A$ that assigns to each variable of Q a *nonempty* subset of A. A valuation for Q is a total function $\theta : \mathit{Var}(Q) \to A$.

Let $\mathcal{A}$ be a relational structure of unary and binary relations. A pre-valuation Θ is called *arc-consistent*[2] iff

- for each unary atom $P(x)$ in Q and each $v \in \Theta(x)$, $P(v)$ is true (in $\mathcal{A}$) and
- for each binary atom $R(x, y)$ in Q, for each $v \in \Theta(x)$ there exists $w \in \Theta(y)$ such that $R(v, w)$ is true and for each $w \in \Theta(y)$ there exists $v \in \Theta(x)$ such that $R(v, w)$ is true.

A valuation θ is called *consistent* if it satisfies the query. In this case, for a Boolean query, we also say that the structure is a *model* of the query and the valuation a *satisfaction*. Obviously, a valuation is consistent if and only if the pre-valuation Θ defined by $\Theta(x) \mapsto \{\theta(x)\}$ is arc-consistent.

Example 6.1 Consider the Boolean conjunctive query $q \leftarrow R(x, y), S(x, y)$ and the database

R		
	1	2
	3	4

S		
	3	2
	1	4

Then $\Theta : x \mapsto \{1, 3\}, y \mapsto \{2, 4\}$ is an arc-consistent prevaluation of q. However, q is not consistent. □

Proposition 6.2 (Folklore) *There is an algorithm which checks in time* $O(||\mathcal{A}|| \cdot |Q|)$ *whether an arc-consistent pre-valuation of* Q *on* $\mathcal{A}$ *exists, and if it does, returns one.*

[2]This notion is well-known in constraint satisfaction, cf. [21].

Proof. We phrase the problem of computing Θ by deciding, for each x, v, whether $v \notin \Theta(x)$ as an instance $\mathcal{P}$ of propositional Horn-SAT. The propositional predicates are the atoms $\overline{\Theta}(x, v)$ (where $x \in \mathit{Vars}(Q)$, $v \in A$ are constants), and the Horn clauses are

$\{\overline{\Theta}(x, v) \leftarrow . \mid P(x) \in Q,\ v \in A,\ \neg P^{\mathcal{A}}(v)\} \cup$
$\{\overline{\Theta}(x, v) \leftarrow \bigwedge\{\overline{\Theta}(y, w) \mid R^{\mathcal{A}}(v, w)\}. \mid R(x, y) \in Q,\ v \in A\} \cup$
$\{\overline{\Theta}(y, w) \leftarrow \bigwedge\{\overline{\Theta}(x, v) \mid R^{\mathcal{A}}(v, w)\}. \mid R(x, y) \in Q,\ w \in A\}$

Let $\overline{\Theta}$ be the binary relation defined by $\mathcal{P}$ and let

$$\Theta(x) \mapsto \{v \in A \mid (x, v) \notin \overline{\Theta}\}.$$

If $\Theta(x) = \emptyset$ for some variable x, no arc-consistent pre-valuation of Q on $\mathcal{A}$ exists and Q is not satisfied. Otherwise, Θ is an arc-consistent pre-valuation and contains all arc-consistent pre-valuations of Q and $\mathcal{A}$.

We can compute program $\mathcal{P}$, evaluate it using Minoux' algorithm ([59], see Figure 3), and compute Θ in total time linear in the size of $\mathcal{P}$, which is $O(||\mathcal{A}|| \cdot |Q|)$. □

Actually, this algorithm computes the unique subset-maximal arc-consistent pre-valuation of Q on $\mathcal{A}$.

Let $<$ be a total order on $A = |\mathcal{A}|$ and Θ be a pre-valuation. Then the valuation θ with $\theta(x) \mapsto v$ iff v is the smallest node in $\Theta(x)$ w.r.t. $<$ is called the *minimum valuation* w.r.t. $<$ in Θ.

Definition 6.3 Let $\mathcal{A}$ be a relational structure, R a binary relation in $\mathcal{A}$, and $<$ a total order on $A = |\mathcal{A}|$. Then, R is said to have the *$\underline{\text{X}}$-property w.r.t. $<$* iff for all $n_0, n_1, n_2, n_3 \in A$ such that $n_0 < n_1$ and $n_2 < n_3$,

$$R(n_1, n_2) \wedge R(n_0, n_3) \Rightarrow R(n_0, n_2).$$

Figure 5 illustrates why the property is called $\underline{\text{X}}$ (read as "X-underbar"). Let us consider two vertical bars both representing the order $<$ bottom-up (i.e., with the smallest value at the bottom). Let each edge (u, v) in R be represented by an arc from node u on the left bar to node v on the right bar. Then, whenever there are two crossing arcs (u, v) and (u', v') in this diagram, then there must be an arc $(\min(u, u'), \min(v, v'))$, the "underbar", in the diagram as well.

The $\underline{\text{X}}$-property was introduced in [45]. It was shown there that the H-coloring problem (or equivalently the conjunctive query evaluation problem) on graphs H with the $\underline{\text{X}}$-property is polynomial-time solvable (see also [46]). We rephrase this result as a tool for efficiently evaluating conjunctive queries.

Let $\mathcal{A}$ be a structure of unary and binary relations and let $<$ be a total order on $|\mathcal{A}|$. Structure $\mathcal{A}$ is said to have the $\underline{\text{X}}$-property w.r.t. $<$ if all binary relations R in $\mathcal{A}$ have the $\underline{\text{X}}$-property w.r.t. $<$.

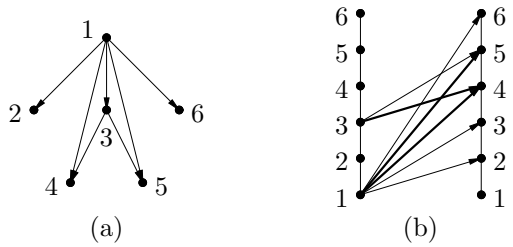


Figure 5: The X-property. Graph (a) and its illustration by arcs between two bars (b). For crossing arcs $R(u, v)$ and $R(u', v')$, say $u < u'$ and $v' < v$, there must be an arc $R(u, v')$.

Lemma 6.4 *Let $\mathcal{A}$ be a structure with the X-property w.r.t. $<$ and let Θ be an arc-consistent pre-valuation on $\mathcal{A}$ for a given conjunctive query over the relations of $\mathcal{A}$. Then, the minimum valuation in Θ w.r.t. $<$ is consistent.*

Proof. Let θ denote the minimum valuation in Θ w.r.t. $<$. To prove θ consistent, we only need to show the following: If $R(x, y)$ is any binary atom of Q with variables x, y then $R(\theta(x), \theta(y))$ is true in $\mathcal{A}$. Let $\theta(x) = n_0$ and $\theta(y) = n_2$. Since Θ is arc-consistent there exists a node $n_1 \in \Theta(x)$ such that $R(n_1, n_2)$ and a node $n_3 \in \Theta(y)$ such that $R(n_0, n_3)$. If $n_0 = n_1$ or $n_2 = n_3$ then $R(\theta(x), \theta(y)) = R(n_0, n_2)$ is true and we are done. Otherwise, since θ is a minimum valuation we have $n_0 < n_1$ (because $n_0 = \theta(x) = \min \Theta(x)$, $n_1 \in \Theta(x)$, and $n_0 \neq n_1$) and $n_2 < n_3$ (because $n_2 = \theta(y) = \min \Theta(y)$, $n_3 \in \Theta(y)$, and $n_2 \neq n_3$). Then it follows from Definition 6.3 that $R(n_0, n_2)$. □

Clearly, if no arc-consistent pre-valuation of Q on $\mathcal{A}$ exists, there is no consistent valuation for Q on $\mathcal{A}$.

Theorem 6.5 (GWW1992) *Given a structure $\mathcal{A}$ with the X-property w.r.t. $<$ and a Boolean conjunctive query Q over $\mathcal{A}$, Q can be evaluated on $\mathcal{A}$ in time $O(||\mathcal{A}|| \cdot |Q|)$.*

Proof. By Lemma 6.4, all we need to do to check whether a Boolean query Q is satisfied is to try to compute the subset-maximal arc-consistent pre-valuation Θ with respect to Q. By Proposition 6.2, this can be done in time $O(||\mathcal{A}|| \cdot |Q|)$. If it exists, Q returns true; otherwise, Q returns false. □

If follows that checking whether a given tuple $\langle a_1, \ldots, a_k \rangle$ is in the result of a k-ary conjunctive query on structures with the X-property w.r.t. some order can be decided in time $O(||\mathcal{A}|| \cdot |Q|)$ as well. All we need to do is to add (new) singleton unary relations $X_1 = \{a_1\}, \ldots, X_k = \{a_k\}$ to $\mathcal{A}$ and to rewrite the query $Q(x_1, \ldots, x_k) \leftarrow \Phi(x_1, \ldots, x_k)$ into the Boolean query $Q \leftarrow \Phi(x_1, \ldots, x_k) \wedge$

$X_1(x_1) \wedge \cdots \wedge X_k(x_k)$. A k-ary conjunctive query Q over $\mathcal{A}$ with $A = |\mathcal{A}|$ can thus be evaluated on $\mathcal{A}$ in time $O(|A|^k \cdot ||\mathcal{A}|| \cdot |Q|)$.

It is straightforward to show that

Proposition 6.6 ([30]) *The axes*

1. *Child$^+$ and Child* have the $\underline{X}$-property w.r.t. $<_{pre}$,*
2. *Following has the $\underline{X}$-property w.r.t. $<_{post}$, and*
3. *Child, NextSibling, NextSibling*, and NextSibling$^+$ have the $\underline{X}$-property w.r.t. $<_{bflr}$.*

It follows immediately from Theorem 6.5 that

Corollary 6.7 *Conjunctive queries over each of the following signatures are in polynomial time w.r.t. combined complexity:*

$$\begin{aligned}
\tau_1 &:= \langle (Lab_a)_{a\in\Sigma}, Child^+, Child^*\rangle \\
\tau_2 &:= \langle (Lab_a)_{a\in\Sigma}, Following\rangle \\
\tau_3 &:= \langle (Lab_a)_{a\in\Sigma}, Child, NextSibling, \\
&\qquad NextSibling^*, NextSibling^+\rangle
\end{aligned}$$

One can verify that Proposition 6.6 lists all the cases where the $\underline{X}$-property holds for any of the axis relations and the orders $<_{pre}$, $<_{post}$, or $<_{bflr}$. And indeed, the conjunctive queries over any signature of unary and axis relations not contained in either τ_1, τ_2, or τ_3 are NP-complete: The $\underline{X}$-property yields a complete characterization of the tractability frontier for classes of conjunctive queries over trees given by unary and axis relations.

Theorem 6.8 (Dichotomy Theorem, [35]) *Unless $P = NP$, the conjunctive queries over structures of binary axis relations F and unary relations are in P if and only if there is a total order $<$ such that all binary relations in F have the $\underline{X}$-property w.r.t. $<$. For each such class of queries, the query evaluation problem is either in P or NP-complete.*

A precise complexity characterization within PTIME of the polynomial classes of queries remains open.

Holistic Processing of Acyclic Queries

The maximal arc-consistent pre-valuation Θ of a query as computed by the algorithm of Proposition 6.2 subsumes all consistent valuations of θ. If $\theta(x) = v$, then $v \in \Theta(x)$. For acyclic conjunctive queries, the converse holds as well.

algorithm enumerate_satisfactions(int i)
// Let the variables of query Q be numbered $x_1, \ldots, x_n$
// by a pre-order depth-first left-to-right traversal of the
// query tree. Let parent(x_i) denote the parent variable
// of x_i in the query tree. Partial function θ is a global
// variable.
begin
  for each $v \in \Theta(x_i)$ **do**
  if $i = 1$ or tuple $(\theta(\text{parent}(x_i)), v)$ satisfies the atom
    of Q connecting parent(x_i) with x_i **then**
  begin
    $\theta(x_i) := v$;
    if $i = n$ **then** output θ;
    else enumerate_satisfactions(i+1);
  end
end.

Figure 6: Algorithm for enumerating solutions of an acyclic conjunctive query Q represented by an arc-consistent pre-valuation Θ.

Proposition 6.9 *Let Q be an acyclic conjunctive query over unary and binary relations and let Θ be an arc-consistent pre-valuation of Q. Then for any variable x of Q and each $v \in \Theta(x)$, there is a consistent valuation θ of Q such that $\theta(x) = v$.*

Proof. We may assume that Q is connected (if this is not the case, we can connect the query by adding atoms of the complete binary relation $A \times A$, with A the domain of the structure). By Q_y, we denote the subset of atoms of Q in which y either appears or which are below y in the query tree. Without loss of generality, Q is oriented in such a way that variable x is the root and Q_x is Q.

We prove by induction that for any variable y and any arc-consistent pre-valuation Θ of Q_y with $u \in \Theta(y)$, there is a consistent valuation θ of Q_y such that $\theta(y) = u$.

Induction start: If y is a leaf of the query tree, Q_y consists only of unary atoms over y. Clearly, $u \in \Theta(y)$ implies that $\theta : y \mapsto u$ is consistent.

Induction step: Let the variables $y_1, \ldots, y_n$ be the children of y in the query tree. Let there be an arc-consistent pre-valuation Θ of Q_y with $u \in \Theta(y)$. Then there must be data tree nodes $u_1, \ldots, u_n$ such that for each i the binary atom $R(y, y_i)$ (or $R(y_i, y)$) that connects y and y_i holds on (u, u_i) resp. (u_i, u). But then the restriction $\Theta|_{Q_{y_i}}$ of Θ to the variables of Q_{y_i} is an arc-consistent pre-valuation of Q_{y_i} with $u_i \in \Theta|_{Q_{y_i}}(y_i)$. By the induction hypothesis, there is a consistent valuation θ_{y_i} of Q_{y_i} with $\theta_{y_i}(y_i) = u_i$. But then the valuation θ with $y \mapsto u$ and $z \mapsto \theta_{y_i}(z)$ for each variable z of Q_{y_i} and each $1 \leq i \leq n$ is consistent

for Q_y. □

But then the maximum arc-consistent pre-valuation for a query and a structure computed by the algorithm of Proposition 6.2 is also a compact representation of precisely all the solutions of Q. These solutions can be read out from the pre-valuation by the recursive algorithm shown in Figure 6, invoked as enumerate_satisfactions(1).

Note that Proposition 6.9 is a rephrasing of a known result about acyclic queries: each tuple in the result of a *full reducer* contributes to a valuation [77] (cf. [1]). Computing the maximal arc-consistent prevaluation for an acyclic query is nothing other than applying a full reducer.

Proposition 6.9 guarantees that the algorithm of Figure 6 does not need to perform any backtracking to enumerate all solutions. Given Θ, enumerate_satisfactions(1) runs in time $O(|A| \cdot ||Q(\mathcal{A})||)$, where A is the domain of the structure and $||Q(\mathcal{A})||$ is the size of the output of the query.

This approach has been taken in work on so-called holistic twig joins [13, 48]. Indeed, this approach to evaluating a tree-shaped query does not proceed step by step but evaluates all structural joins at once. In [13], it is shown how path-shaped queries that only involve *Child* and *Child*$^+$ as binary relations can be processed efficiently. As we have seen, this approach can be generalized much beyond path queries and these two axes.

The algorithms of [13] also suggest further structure within the sets $\Theta(x)$ and the use of pointers between the elements of $\Theta(y)$ and $\Theta(y_i)$ if y_i is a child of y in the query tree. This allows to iterate only over elements of the Θ that are relevant to solutions and to reduce the running time of enumerate_satisfactions(1) to time $O(||Q(\mathcal{A})||)$. For simplicity, let us assume a uniform machine model in which pointers have constant size. Then,

Proposition 6.10 *The k-ary acyclic conjunctive queries Q on trees $\mathcal{A}$ can be evaluated in time $O(|Q| \cdot ||\mathcal{A}|| + ||Q(\mathcal{A})||)$.*

7 Complexity Summary

In this section I provide a summary of previous results on the complexity of queries on trees. In order not to be overly repetitive, I refer to Section 4 for results on MSO, FO, and FO^k and Section 6 for results on conjunctive queries.

An overview of the complexity results discussed in this section can be found in Figure 7. These are put into context with a display of the relative expressiveness of the query languages. The Venn diagram notation refers to complexity classes (we make the usual complexity-theoretic assumptions that $LOGCFL \subset P \subset NP \subset PSPACE$) and the arrows refer to expressive power; $L_1 \to L_2$ means that each query in language L_1 can be translated into an equivalent query in L_2. The

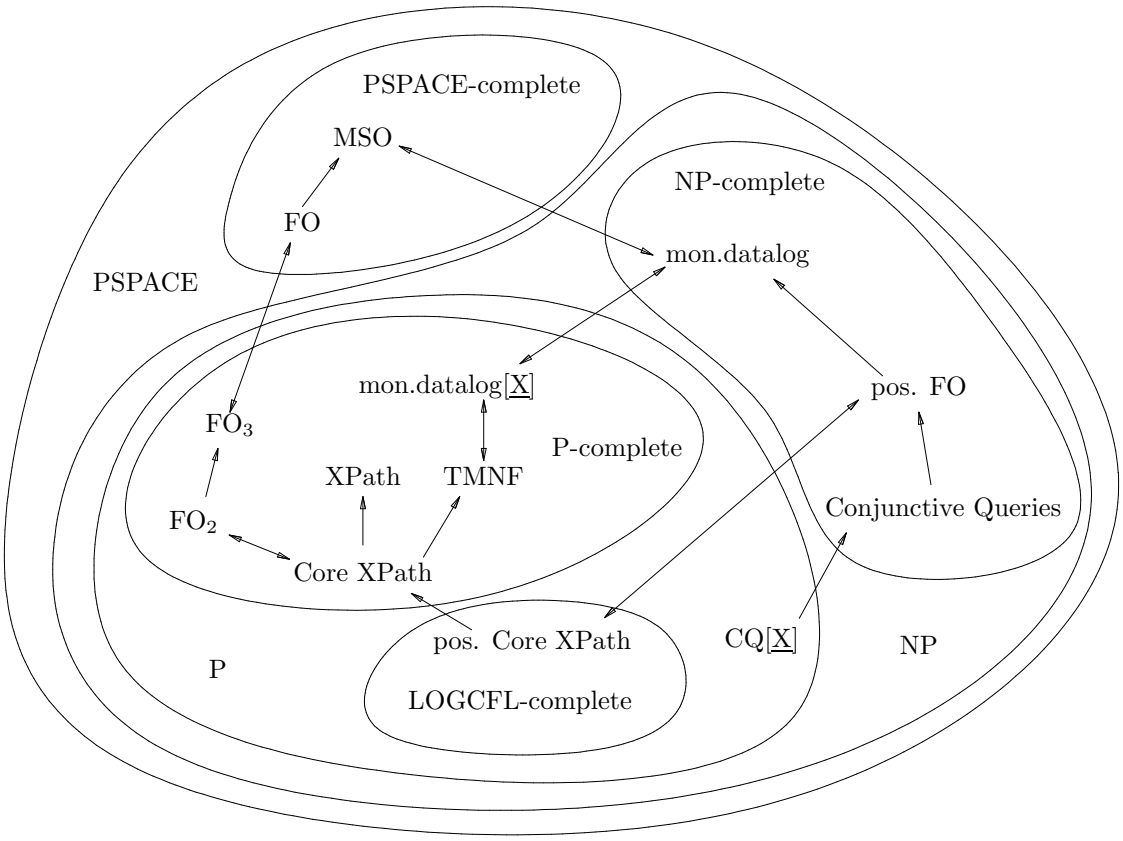


Figure 7: Complexity and expressive power of query languages over trees.

notation $L[\underline{X}]$ refers to the queries of language L using only binary relations from an axis set for which conjunctive queries are tractable by the $\underline{X}$-property (see Section 6).

Monadic Datalog

As observed in Section 3, while full datalog is EXPTIME-complete (c.f. e.g. to [20]), monadic datalog over arbitrary finite structures is in NP. The complexity of monadic datalog over a given set of axes is always the same as that of conjunctive queries over the same axes (see Section 6). Monadic datalog over trees defined by unary relations and the binary relations *FirstChild* and *NextSibling* is P-complete [31] and can be solved in time linear in the size of the database and linear in the size of the tree. In the case that all individual rules are acyclic (conjunctive queries), it is known from [31] that monadic datalog over arbitrary axes can be evaluated in linear time.

XPath

In [33], it is shown that XPath 1 is in PTIME w.r.t. combined complexity. This result is originally obtained through a dynamic programming algorithm [33]. However, the dynamic programming algorithm computes many useless intermediate results and consumes much memory. To fix this, a more efficient top-down algorithm is given in [33] as well. This algorithm still runs in polynomial time, with better worst-case upper bounds on running time and memory consumption (namely time $O(|A|^3 \cdot ||\mathcal{A}|| \cdot |Q|^2)$ and space $O(|A| \cdot ||\mathcal{A}|| \cdot |Q|^2)$). Further work on polynomial-time algorithms for full XPath 1 which elaborates on the results of [33] and integrates them into a native XML database management system can be found in [12]. This work also shows how to integrate XQuery and efficient XPath processing using a single native algebra.

Positive Core XPath is complete for LOGCFL, a parallel complexity class in NC_2 (combined complexity). In [34], LogCFL membership is proven for a much larger fragment of XPath than Core XPath which excludes only a very small number of features apart from negation from full XPath – it even supports arithmetics and aggregations. Unfortunately, the positive result on the parallel complexity of positive XPath does not even extend to Core XPath (with negation): Core XPath is PTime-hard (combined complexity) [34].

This PTime-hardness result essentially depends on the presence of transitive axes: Core XPath using only the *Child*, $Child^{-1}$, *NextSibling*, and $NextSibling^{-1}$ axes is in LogSpace w.r.t. combined complexity [34].

The precise complexity of Core XPath with only forward or only downward axes (*Child* and $Child^{+}$) remains open, even though these are fragments that have important applications in the context of tree pattern matching and stream processing.

The data complexity of XPath depends on encodings. XPath 1.0 on DOM trees (pointer structures) is LogSpace-complete if the concatenation operation on strings and multiplication are excluded from the language.

So far, we have always assumed that the input is basically given as a pointer structure. But XML documents can also be considered in their natural textual (string) representation. The distinction is only relevant for the very small complexity classes inside LogSpace, for which completeness is usually defined in terms of reductions not strong enough to map between DOM trees and strings. On string representations, Core XPath was shown to be in TC^0 [34], a complexity class inside LogSpace.

The query complexity of XPath 1.0 is in LogSpace [33]. This is a slightly curious fact. While for virtually all known traditional query languages, the query complexity is greater than the data complexity by at least an exponential factor (cf. e.g. [1]), this is not the case of XPath.

The PTime-hardness of Core XPath suggests (but does not prove) [39] that any query evaluation algorithm has to consume at least linear amounts of memory.

For the more restrictive stream processing scenario, this can be proven: It has been shown in [40] that any streaming algorithm for the Boolean Core XPath queries must consume memory at least linear in the depth of the data tree. Of course, there are trees whose depth is linear in their size, so one can interpret this result in the sense that there can be no streaming algorithm for Core XPath that takes space less than linear in the *size* of the XML stream. Memory-efficient – and thus scalable – stream processing for XPath is, from a certain point of view, in the worst case impossible. Fortunately, XML trees tend to be shallow in practice, so showing this lower bound tight can be considered a positive result. And indeed, the following is implicit in [60, 70]: Let $\mathbf{T}$ be a tree-language. If $\mathbf{T}$ is definable by an MSO-sentence, then $\mathbf{T}$ can be recognized by a streaming algorithm using memory $O(\mathit{depth}(\cdot))$. It follows that there is a streaming algorithm for Boolean Core XPath with memory consumption $O(\mathit{depth}(\cdot))$.

Acknowledgments

I am grateful to my coauthors from [29, 33, 34, 35, 10], Michael Benedikt, Markus Frick, Georg Gottlob, Martin Grohe, Reinhard Pichler, Klaus Schulz, and Luc Segoufin.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava. "Structural Joins: A Primitive for Efficient XML Query Pattern Matching". In *18th International Conference on Data Engineering (ICDE'02)*, 2002.

[3] M. Altinel and M. Franklin. "Efficient Filtering of XML Documents for Selective Dissemination of Information". In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB'2000)*, pages 53–64, Cairo, Egypt, 2000.

[4] M. Arenas and L. Libkin. "XML data exchange: consistency and query answering". In *Proc. PODS 2005*, pages 13–24, 20005.

[5] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. "On the Memory Requirements of XPath Evaluation over XML Streams". In *Proceedings of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'04)*, pages 177–188, 2004.

[6] R. Baumgartner, S. Flesca, and G. Gottlob. "Visual Web Information Extraction with Lixto". In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*, pages 119–128, Rome, Italy, 2001.

[7] M. Benedikt. "An Insider's Guide to Logic in Telecommunications Data". In *Proc. LICS 2005*, pages 104–105, 2005.

[8] M. Benedikt, W. Fan, and G. Kuper. "Structural Properties of XPath Fragments". In *Proc. of the 9th International Conference on Database Theory (ICDT)*, pages 79–95, Siena, Italy, 2003.

[9] M. Benedikt and C. Koch. "XPath Leashed", 2006. Submitted for publication.

[10] M. Benedikt and C. Koch. "XPath with Data Values Revisited", 2006. Unpublished manuscript.

[11] S. Bird, Y. Chen, S. Davidson, H. Lee, and Y. Zheng. "Extending XPath to Support Linguistic Queries". In *PLAN-X*, 2005.

[12] M. Brantner, S. Helmer, C.-C. Kanne, and G. Moerkotte. "Full-fledged Algebraic XPath Processing in Natix". In *Proceedings of the 21st IEEE International Conference on Data Engineering (ICDE)*, 2005.

[13] N. Bruno, D. Srivastava, and N. Koudas. "Holistic Twig Joins: Optimal XML Pattern Matching". In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD'02)*, Madison, Wisconsin, June 2002.

[14] P. Buneman, B. Choi, W. Fan, R. Hutchison, R. Mann, and S. Viglas. "Vectorizing and Querying Large XML Repositories". In *Proc. ICDE 2005*, pages 261–272, 2005.

[15] P. Buneman, S. Khanna, K. Tajima, and W. C. Tan. "Archiving Scientific Data". *ACM Transactions on Database Systems*, **29**:2–42, 2004.

[16] C. Y. Chan, P. Felber, M. N. Garofalakis, and R. Rastogi. Efficient Filtering of XML Documents with XPath Expressions. In *Proceedings of the 18th IEEE International Conference on Data Engineering (ICDE)*, San Jose, California, USA, February 26-March 1, 2002, 2000.

[17] C. Chekuri and A. Rajaraman. Conjunctive Query Containment Revisited". In *Proc. of the 6th International Conference on Database Theory (ICDT)*, pages 56–70, Delphi, Greece, 1997.

[18] S. Cosmadakis, H. Gaifman, P. Kanellakis, and M. Vardi. "Decidable Optimization Problems for Database Logic Programs". In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 477–490, Chicago, Illinois, USA, 1988. ACM Press, New York, NY, USA.

[19] B. Courcelle. "Graph Rewriting: An Algebraic and Logic Approach". In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 2, chapter 5, pages 193–242. Elsevier Science Publishers B.V., 1990.

[20] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. "Complexity and Expressive Power of Logic Programming". *ACM Computing Surveys*, **33**(3):374–425, Sept. 2001.

[21] R. Dechter. *"Constraint Processing"*. Morgan Kaufmann, May 2003.

[22] A. Deutsch and V. Tannen. "MARS: A System for Publishing XML from Mixed and Redundant Storage". In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, pages 201–212, Berlin, Germany, 2003.

[23] P. Dietz and D. Sleator. "Two algorithms for maintaining order in a list". In *Proc. 19th Annual ACM Symp. Theory of Computing (STOC)*, pages 365–372, 1987.

[24] J. Doner. "Tree Acceptors and some of their Applications". *Journal of Computer and System Sciences*, **4**:406–451, 1970.

[25] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 1999. Second edition.

[26] M. Fernandez, D. Florescu, J. Kang, and A. Levy. "Catching the Boat with Strudel: Experiences with a Web-Site Management System". In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD'98)*, pages 414–425, Seattle, WA USA, June 1998.

[27] T. Fiebig and G. Moerkotte. "Evaluating Queries on Structure with eXtended Access Support Relations". In *Proc. WebDB*, 2000.

[28] J. Flum, M. Frick, and M. Grohe. "Query Evaluation via Tree-Decompositions". *Journal of the ACM*, **49**(6):716–752, 2002.

[29] M. Frick, M. Grohe, and C. Koch. "Query Evaluation on Compressed Trees". In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS)*, Ottawa, Canada, June 2003.

[30] G. Gottlob and C. Koch. "Monadic Queries over Tree-Structured Data". In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 189–202, Copenhagen, Denmark, July 2002.

[31] G. Gottlob and C. Koch. "Monadic Datalog and the Expressive Power of Web Information Extraction Languages". *Journal of the ACM*, **51**(1):74–113, 2004.

[32] G. Gottlob, C. Koch, and R. Pichler. "Efficient Algorithms for Processing XPath Queries". In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, pages 95–106, Hong Kong, China, 2002.

[33] G. Gottlob, C. Koch, and R. Pichler. "Efficient Algorithms for Processing XPath Queries". *ACM Transactions on Database Systems*, **30**(2):444–491, June 2005.

[34] G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. "The Complexity of XPath Query Evaluation and XML Typing". *Journal of the ACM*, **52**(2):284–335, Mar. 2005.

[35] G. Gottlob, C. Koch, and K. U. Schulz. "Conjunctive Queries over Trees". *Journal of the ACM*, **53**(2), Mar. 2006.

[36] G. Gottlob, N. Leone, and F. Scarcello. "The Complexity of Acyclic Conjunctive Queries". *Journal of the ACM*, **48**(1):431–498, 2001.

[37] G. Gottlob, N. Leone, and F. Scarcello. "Hypertree Decompositions and Tractable Queries". *Journal of Computer and System Sciences*, **64**(3):579–627, 2002.

[38] T. J. Green, G. Miklau, M. Onizuka, and D. Suciu. "Processing XML Streams with Deterministic Automata". In *Proc. of the 9th International Conference on Database Theory (ICDT)*, 2003.

[39] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, 1995.

[40] M. Grohe, C. Koch, and N. Schweikardt. "Tight Lower Bounds for Query Processing on Streaming and External Memory Data". In *Proc. ICALP*, 2005.

[41] M. Grohe and N. Schweikardt. "Comparing the Succinctness of Monadic Query Languages over Finite Trees". In *Proc. CSL*, pages 226–240, 2003.

[42] T. Grust, S. Sakr, and J. Teubner. "XQuery on SQL Hosts". In *Proc. VLDB 2004*, pages 252–263, 2004.

[43] T. Grust, M. van Keulen, and J. Teubner. "Accelerating XPath Evaluation in any RDBMS". *ACM Transactions on Database Systems*, **29**:91–131, 2004.

[44] A. K. Gupta and D. Suciu. "Stream Processing of XPath Queries with Predicates". In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD'03)*, pages 419–430, 2003.

[45] W. Gutjahr, E. Welzl, and G. Woeginger. "Polynomial Graph Colourings". *Discrete Applied Math.*, **35**:29–46, 1992.

[46] P. Hell and J. Nesetril. *Graphs and Homomorphisms.* Oxford University Press, 2004.

[47] N. Immerman. *"Descriptive Complexity"*. Springer Graduate Texts in Computer Science, 1999.

[48] H. Jiang, W. Wang, and H. Lu. "Holistic Twig Joins on Indexed XML Documents". In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, 2003.

[49] D. S. Johnson. "A Catalog of Complexity Classes". In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 1, chapter 2, pages 67–161. Elsevier Science Publishers B.V., 1990.

[50] V. Josifovski and M. F. Fontoura. "Querying XML Streams". *VLDB Journal*, **14**(2):197–210, April 2005.

[51] C. Koch. "Efficient Processing of Expressive Node-Selecting Queries on XML Data in Secondary Storage: A Tree Automata-based Approach". In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, pages 249–260, 2003.

[52] C. Koch and S. Scherzinger. "Attribute Grammars for Scalable Query Processing on XML Streams". *VLDB Journal*, 2006. to appear.

[53] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier. "Schema-based Scheduling of Event Processors and Buffer Minimization for Queries on Structured Data Streams". In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, Toronto, Canada, 2004.

[54] P. Kolaitis and M. Vardi. "Conjunctive Query Containment and Constraint Satisfaction". *Journal of Computer and System Sciences*, **61**(2):302–332, 2000.

[55] L. Libkin. *Elements of Finite Model Theory.* Springer, 2004.

[56] Library of Congress. "MARCXML – MARC 21 XML Schema", 2005. http://www.loc.gov/standards/marcxml/.

[57] M. Marx. "First Order Paths in Ordered Trees". In *Proc. of the 10th International Conference on Database Theory (ICDT)*, pages 114–128, 2005.

[58] A. R. Meyer. "Weak Monadic Second Order Theory of Successor is not Elementary-Recursive". In *Logic Colloquium, Lecture Notes in Mathematics 453*, pages 132–154. Springer-Verlag, N.Y., 1975.

[59] M. Minoux. "LTUR: A Simplified Linear-Time Unit Resolution Algorithm for Horn Formulae and Computer Implementation". *Information Processing Letters*, **29**(1):1–12, 1988.

[60] A. Neumann and H. Seidl. "Locating Matches of Tree Patterns in Forests". In *Proc. 18th FSTTCS, LNCS 1530*, pages 134–145, 1998.

[61] D. Olteanu, T. Kiesling, and F. Bry. "An Evaluation of Regular Path Expressions with Qualifiers against XML Streams". In *Proceedings of 19th International Conference on Data Engineering (ICDE)*, Bangalore, India, 5th - 8th March 2003. Full version in Technical Report PMS-FB-2002-12, Ludwig-Maximilians-Uniiversität München, Munich, Germany, 2002.

[62] D. Olteanu, H. Meuss, T. Furche, and F. Bry. "XPath: Looking Forward". In *Proc. EDBT Workshop on XML Data Management*, volume LNCS 2490, pages 109–127, Prague, Czech Republic, 2002. Springer-Verlag.

[63] P. E. O'Neil, E. J. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. "ORDPATHs: Insert-Friendly XML Node Labels". In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD'04)*, pages 903–908, 2004.

[64] Y. Papakonstantinou, V. R. Borkar, M. Orgiyan, K. Stathatos, L. Suta, V. Vassalos, and P. Velikhov. "XML queries and algebra in the Enosys integration platform". *Data Knowl. Eng.*, **44**(3):299–322, 2003.

[65] F. Peng and S. Chawathe. "XPath Queries on Streaming Data". In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD'03)*, 2003.

[66] P. Rao and B. Moon. "PRIX: Indexing And Querying XML Using Prüfer Sequences". In *Proceedings of the 20th IEEE International Conference on Data Engineering (ICDE)*, pages 288–300, 2004.

[67] K. Reinhardt. "The Complexity of Translating Logic to Finite Automata". In E. Grädel, W. Thomas, and T. Wilke, editors, *Automata, Logics, and*

Infinite Games – A Guide to Current Research. Springer-Verlag, LNCS 2500, 2002.

[68] A. Sahuguet and F. Azavant. "Building Intelligent Web Applications Using Lightweight Wrappers". *Data and Knowledge Engineering*, **36**(3):283–316, 2001.

[69] T. Schwentick. "On Diving in Trees". In *Proc. International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 660–669, 2000.

[70] L. Segoufin and V. Vianu. "Validating Streaming XML Documents". In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'02)*, 2002.

[71] J. Thatcher and J. Wright. "Generalized Finite Automata Theory with an Application to a Decision Problem of Second-order Logic". *Mathematical Systems Theory*, **2**(1):57–81, 1968.

[72] J. Thierry-Mieg and R. Durbin. "Syntactic Definitions for the ACeDB Data Base Manager". Technical Report MRC-LMB xx.92, MRC Laboratory for Molecular Biology, Cambridge, UK, 1992.

[73] M. Y. Vardi. "The Complexity of Relational Query Languages". In *Proc. 14th Annual ACM Symposium on Theory of Computing (STOC'82)*, pages 137–146, San Francisco, CA USA, May 1982.

[74] H. Wang, S. Park, W. Fan, and P. S. Yu. "ViST: A Dynamic Index Method for Querying XML Data by Tree Structures". In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD'03)*, pages 110–121, 2003.

[75] F. Weigel, K. U. Schulz, and H. Meuss. "The BIRD Numbering Scheme for XML and Tree Databases - Deciding and Reconstructing Tree Relations Using Efficient Arithmetic Operations". In *Proc. XSym 2005*, pages 49–67, 2005.

[76] World Wide Web Consortium. XML Path Language (XPath) Recommendation. http://www.w3c.org/TR/xpath/, Nov. 1999.

[77] M. Yannakakis. "Algorithms for Acyclic Database Schemes". In *Proceedings of the 7th International Conference on Very Large Data Bases (VLDB'81)*, pages 82–94, Cannes, France, 1981.