# Satisfiability of XPath Queries with Sibling Axes

Floris Geerts[1] and Wenfei Fan[2]

[1] Hasselt University and University of Edinburgh
[2] University of Edinburgh and Bell Laboratories

**Abstract.** We study the satisfiability problem for XPath fragments supporting the following-sibling and preceding-sibling axes. Although this problem was recently studied for XPath fragments without sibling axes, little is known about the impact of the sibling axes on the satisfiability analysis. To this end we revisit the satisfiability problem for a variety of XPath fragments with sibling axes, in the presence of DTDs, in the absence of DTDs, and under various restricted DTDs. In these settings we establish complexity bounds ranging from NLOGSPACE to undecidable. Our main conclusion is that in many cases, the presence of sibling axes complicates the satisfiability analysis. Indeed, we show that there are XPath satisfiability problems that are in PTIME and PSPACE in the absence of sibling axes, but that become NP-hard and EXPTIME-hard, respectively, when sibling axes are used instead of the corresponding vertical modalities (e.g., the wildcard and the descendant axis).

## 1 Introduction

We revisit the *satisfiability problem* for XPath [7] in the presence of DTDs. It is the problem to determine, given an XPath query $Q$ and a DTD $D$, whether or not there exists an XML document $T$ such that $T$ conforms to $D$ and satisfies $Q$, i.e., the set $Q(T)$ of nodes of $T$ selected by $Q$ is nonempty.

The prevalent use of XPath highlights the need for the satisfiability analysis of XPath queries. Indeed, XPath has been commonly used in specifying XML constraints (e.g., [6, 9, 27]), queries (e.g., XSLT, XQuery), updates (e.g., [26]), and access control (e.g., [10]). In many applications both XPath expressions and DTDs are present. The static satisfiability analysis of XPath addresses the interaction between XPath and DTDs, and is useful in query optimization, update manipulation and reasoning about XML access control, among other things. An alternative to the static analysis would be a dynamic approach. As an example, consider an access-control policy $S$ defined in terms of a DTD and XPath queries, which is to prevent disclosure of XML documents to unauthorized users by validating that the documents "satisfy" $S$. One could simply attempt to validate a document with respect to $S$ at run-time. This, however, would not tell us whether repeated failures are due to inconsistency between the XPath queries and the DTD, or problems with the documents.

The satisfiability problem has been studied for a large number of XPath fragments [2, 13, 15, 17], in the presence and in the absence of DTDs. The previous work has mostly focused on XPath queries with only vertical modalities

G. Bierman and C. Koch (Eds.): DBPL 2005, LNCS 3774, pp. 122–137, 2005.

such as *child*, *parent*, *descendant* and *ancestor* axes (referred to "$\downarrow, \uparrow, \downarrow^*, \uparrow^*$", respectively). However, XML data is ordered and it is often desirable to access this order using XPath. Indeed, consider an XML document storing items bought by customers over a period of time. The items are grouped under customers and appear according to their date of acquisition. In order to detect customer behavior over time, one needs to be able to pose queries involving order. Therefore, it is common to find XPath queries that need sideways traversal via *horizontal* modalities such as (immediate) *right-sibling* and *left-sibling* axes (denoted by "$\rightarrow, \rightarrow^*, \leftarrow, \leftarrow^*$", respectively). It is natural to ask whether the presence of sibling axes simplifies or complicates the satisfiability analysis. For example, consider a fragment $\mathcal{X}(\downarrow, [\,])$ that supports wildcard ($\downarrow$) and qualifiers ($[\,]$) and characterizes well-studied tree pattern queries [1, 2, 28, 29]. One would want to know whether the satisfiability analysis becomes easier or harder for $\mathcal{X}(\rightarrow, [\,])$ (resp. $\mathcal{X}(\leftarrow, [\,])$), the horizontal counterpart of $\mathcal{X}(\downarrow, [\,])$ by substituting $\rightarrow$ (resp. $\leftarrow$) for $\downarrow$. The complexity of the satisfiability analysis is not yet known for a variety of XPath fragments with sibling axes.

Related to the satisfiability analysis is the *containment problem*, which is to determine, given two XPath queries $Q_1, Q_2$ and a DTD $D$, whether or not for all XML documents $T$ that conform to $D$, $Q_1(T)$ is contained in $Q_2(T)$. While there has also been a host of work on the containment analysis [8, 17, 19, 22, 29], the previous results cannot answer the questions of the satisfiability analysis. Indeed, as already observed by [2], the lower bounds for the containment analysis are often much higher than its satisfiability counterpart. Worse still, to our knowledge there has not been a full treatment of the containment problem for various fragments with the sibling axes or XPath negation.

**Main Results.** To this end we investigate the satisfiability problem for a variety of XPath fragments with sibling axes, in the following settings:

- XPath fragments: with or without recursion axis (e.g., $\rightarrow^*, \leftarrow^*, \downarrow^*, \uparrow^*$), qualifiers ($[\,]$), data-value joins (denoted by $=$), and negation ($\neg$);
- DTDs: in the presence of DTDs vs. in the absence of DTDs; fixed DTDs vs. arbitrary DTDs; and restricted DTDs with or without DTD recursion, disjunction, and Kleene star in element type definitions.

We establish lower and upper bounds for the satisfiability analysis in these settings, which range from NLOGSPACE to undecidable. We also explore the impact of sibling axes on the analysis. We show that in the absence of XPath qualifiers, the presence of sibling axes does not complicate the satisfiability analysis. In contrast, in the presence of qualifiers, sibling axes make the analysis harder. Indeed, we show the following. (a) The satisfiability problem for $\mathcal{X}(\rightarrow, [\,])$ is NP-hard under fixed, disjunction-free DTDs, whereas it is in PTIME for its vertical counterpart $\mathcal{X}(\downarrow, [\,])$ in the same setting [2]. (b) It is EXPTIME-hard for $\mathcal{X}(\uparrow, \rightarrow, \cup, [\,], \neg))$, a fragment with upward and sibling axes and negation but without recursion; in contrast, it is in PSPACE for the vertical counterpart $\mathcal{X}(\downarrow, \uparrow, \cup, [\,], \neg))$ [2]. (c) Under non-recursive and fixed DTDs and in the absence of DTDs, it is still unknown [2] whether or not the satisfiability problem is

decidable for $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ], \neg, =)$, a fragment with negation, data-value joins and all the vertical axes. In contrast, the problem is undecidable when sibling axes are introduced; indeed, it is undecidable for $\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\ ], =, \neg)$ in the same settings.

In addition to the complexity bounds for the satisfiability problems, we also explore the connection between vertical and horizontal axes and the connection between the satisfiability and containment analysis, establishing first lower bound results for the containment analysis of XPath fragments with sibling axes.

These results help us understand the interaction between different XPath axes, as well as their interaction with various DTD constructs. Taken together, these results and the previous work [2, 13, 15, 17] provide a detailed treatment of the satisfiability analysis for a large number of XPath fragments commonly found in practice, in a variety of DTD settings.

**Related Work.** The satisfiability problem has been studied in [2, 13, 15, 17]. Complexity bounds were provided in [2] for various XPath fragment under a variety of DTDs. However, no sibling axes were considered there. Our results in this paper complement and extend the results of [2]. The main focus of [17] is about extensions of XPath, and it provided EXPTIME (lower and upper) bounds on equivalence for an extension of XPath in the presence of DTDs, which implies an EXPTIME bound for our fragment with all the axes and negation but without data-value joins. We will show in Section 4.3 that the EXPTIME-hardness already holds for a subclass of the fragment without recursion axes. The XPath queries considered in [15] are basically tree patterns with node equality, inequality and limited use of data joins; neither negation nor sibling axes were considered there; furthermore, DTDs were restricted to be non-recursive disjunction-free in [15]. In the absence of DTDs, [13] studied the satisfiability problem for XPath without negation and data-value joins. From the results of [2], we already know that these bounds do not hold in the presence of DTDs. In particular, [13] gave PTIME bounds for XPath fragments with qualifiers, sibling axes, upward axes, and a root test in the absence of DTDs. We show that in the presence of DTDs, the problem is NP-hard, and we give PTIME bounds in the absence of qualifiers, and in the presence of sibling, upward axes and DTDs.

There has also been work on the containment problem for XPath fragments in the absence and in the presence of DTDs [8, 17, 19, 22, 29]. Most of the work (except [22, 17]) only studied fragments without upward axes, sibling axes, data-value joins and negation. The negation defined in [22] is quite different from the general XPath negation operator. See [25] for a recent survey. As shown in [2], the complexity bounds for the containment analysis are typically much higher than its satisfiability counterpart in the absence of negation. In the presence of negation, the connection between the containment analysis and its satisfiability counterpart was explored in [2] and will be further discussed in Section 5.

Other active areas of XPath research include the expressive power of XPath (e.g., [3, 12, 16, 17, 18, 20, 21]) and query rewriting and minimization (e.g., [1, 9, 11, 23, 28]). While XPath satisfiability is not the focus in those

areas, the satisfiability analysis is useful for XPath rewriting, minimization and optimization.

**Organization.** Section 2 reviews DTDs and defines XPath fragments. Section 3 explores the connection between sibling and vertical axes. Section 4 studies the satisfiability problem for XPath fragments with sibling axes, followed by the containment analysis in Section 5. Section 6 summarizes the main results of the paper. All proofs can be found in the full paper.

## 2 Preliminaries

In this section we first review DTDs [5] and describe the XPath [7] fragments considered in this paper. We then state the satisfiability problem in the presence of DTDs and address its connection with the counterpart in the absence of DTDs.

### 2.1 DTDs

Without loss of generality, we represent a Document Type Definition (DTD [5]) $D$ as ($Ele$, $Att$, $P$, $R$, $r$), where (1) $Ele$ is a finite set of *element types*, ranged over by $A, B, \ldots$; (2) $r$ is a distinguished type in $Ele$, called the *root type*; (3) $P$ is a function that defines the element types: for each $A$ in $Ele$, $P(A)$ is a regular expression over $Ele$; we refer to $A \rightarrow P(A)$ as the *production* of $A$; (4) $Att$ is a finite set of *attribute names*, ranged over by $a, b, \ldots$; and (5) $R$ defines the attributes: for each $A$ in $Ele$, $R(A)$ is a subset of $Att$.

A DTD $D = (Ele, Att, P, R, r)$ is said to be *disjunction-free* if for any element type $A \in Ele$, $P(A)$ does not contain disjunction '+'. It is called *no-star* if for any $A \in Ele$, $P(A)$ does not contain the Kleene star '$*$' (this should not be confused with star-free regular expressions). It is *recursive* if the dependency graph of $D$, which contains an edge $(A, B)$ iff $B$ is in $P(A)$, has a cycle.

An XML document is typically modeled as a (finite) node-labeled tree [5], with nodes additionally annotated with values for attributes. We refer to this as an *XML tree*. An XML tree $T$ *satisfies* (or *conforms to*) a DTD $D = (Ele, Att, P, R, r)$, denoted by $T \models D$, if (1) the root of $T$ is labeled with $r$; (2) each node $n$ in $T$ is labeled with an $Ele$ type $A$, called an $A$ *element*; the label of $n$ is denoted by $\mathsf{lab}(n)$; (3) each $A$ element has a list of *children* such that their labels are a word in the regular language defined by $P(A)$; and (4) for each $A$ in $Ele$ and each $a \in R(A)$, each $A$ element $n$ has a unique $a$ *attribute value*, denoted by $n.a$. We call $T$ an *XML tree of* $D$ if $T \models D$.

*Example 1.* Consider a DTD $D_1 = (Ele, Att, P, R, r)$ defined as

$$Ele = \{r,\ X,\ A,\ B\}.$$
$$P: \quad r \rightarrow X^*, \qquad X \rightarrow (A, B^*)^*$$
$$Att = \emptyset, \quad R(X) = R(T) = R(F) = \emptyset.$$

It is non-recursive and disjunction-free. An XML tree of $D_1$ is shown at the left in Fig. 1.

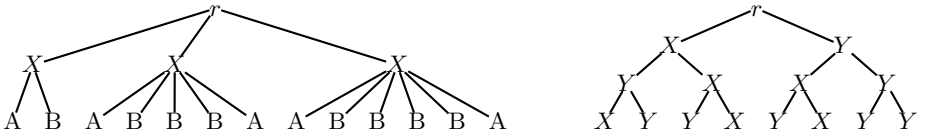


**Fig. 1.** XML trees of the DTDs $D_1$ (left) and $D_2$ (right) given in Example 1

Another DTD $D_2 = (Ele, Att, P, R, r)$ is defined as

$$Ele = \{r,\ X,\ Y\}.$$
$$P: \quad r \to X, Y, \qquad X \to Y, X + \epsilon, \qquad Y \to X, Y + \epsilon$$
$$Att = \emptyset, \quad R(X) = R(T) = R(F) = \emptyset.$$

It is recursive and no-star. An XML tree of $D_2$ is shown at the right in Fig. 1. ■

Note that a DTD $D$ may not have any XML tree $T$ such that $T \models D$. This is because some element type $A$ in $D$ is *non-terminating*, i.e., there exists no finite subtree rooted at an $A$ element that satisfies $D$. Fortunately, one can determine whether this is the case for any element type of $D$ in $O(|D|)$ time, where $|D|$ is the size of $D$ [14]. In the remainder of the paper we will assume that all element types in a DTD are terminating. This does not affect any of our results.

### 2.2 XPath Fragments

Over an XML tree, an XPath query specifies the selection of nodes in the tree. Assume a (possibly infinite) alphabet $\Sigma$ of labels. The largest fragment of XPath studied in this paper, denoted by $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \leftarrow, \leftarrow^*, \rightarrow, \rightarrow^*, \cup, [\ ], =, \neg)$, is defined syntactically as follows:

$$p ::= \epsilon \mid A \mid \downarrow \mid \downarrow^* \mid \uparrow \mid \uparrow^* \mid \rightarrow \mid \rightarrow^* \mid \leftarrow \mid \leftarrow^* \mid p/p \mid p \cup p \mid p[q],$$

where $\epsilon$ and $A$ denote the empty path (the *self-axis*) and a label in $\Sigma$ (the *child-axis*); '$\downarrow$' and '$\downarrow^*$' stand for the wildcard (*child*) and the *descendant-or-self-axis*, while $\uparrow$ and $\uparrow^*$ denote the *parent-axis* and *ancestor-or-self-axis*, respectively; '$\rightarrow^*$' (resp. '$\leftarrow^*$') is the following-sibling (resp. preceding-sibling) axis, and '$\rightarrow$' (resp. '$\leftarrow$') denotes the immediate right sibling (reps. the immediate left sibling); '$/$' and '$\cup$' stand for concatenation and union, respectively; and finally, $q$ in $p[q]$ is called a *qualifier* and is defined by:

$$q ::= p \mid \mathsf{lab}() = A \mid p/@a\ \mathsf{op}\ c \mid p/@a\ \mathsf{op}\ p'/@b \mid q_1 \wedge q_2 \mid q_1 \vee q_2 \mid \neg q,$$

where $p$ is as defined above, $A$ is a label in $\Sigma$, $\mathsf{op}$ is either '$=$' or '$\neq$' (referred to as *data-value joins*), $a, b$ stand for attributes, $c$ is a constant (string value), and $\wedge, \vee, \neg$ stand for *and* (conjunction), *or* (disjunction) and *not* (negation), respectively.

It is worth mentioning that while XPath [7] does not explicitly define '$\leftarrow, \rightarrow$', these operators are definable in terms of the preceding-sibling and following-sibling axes, together with *position()*, as follows:

$$\leftarrow \;=\; \leftarrow^*[position() = 1], \qquad\qquad \rightarrow \;=\; \rightarrow^*[position() = 1].$$

A query $p$ in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \leftarrow, \leftarrow^*, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg)$ over an XML tree $T$ is interpreted as a binary predicate on the nodes of $T$, while a qualifier is interpreted as a unary predicate. More specifically, for any node $n$ in $T$, $T$ *satisfies* $p$ *at* $n$ iff $T \models \exists n'\ p(n, n')$, where $T \models p(n, n')$ and the associated version for qualifiers, $T \models q(n)$, are defined inductively on the structure of $p, q$, as follows:

1. if $p = \epsilon$, then $n = n'$;
2. if $p = l$, then $n'$ is a child of $n$, and is labeled $l$;
3. if $p = \downarrow$, then $n'$ is a child of $n$, regardless of its label;
4. if $p = \downarrow^*$, then $n'$ is either $n$ or a descendant of $n$;
5. if $p = \uparrow$, then $n'$ is the parent of $n$;
6. if $p = \uparrow^*$, then $n'$ is either $n$ or an ancestor of $n$;
7. if $p = \rightarrow$, then $n'$ is the immediate right sibling of $n$.
8. if $p = \rightarrow^*$, then $n'$ is either $n$ or a right sibling of $n$.
9. if $p = \leftarrow$, then $n'$ is the immediate left sibling of $n$.
10. if $p = \leftarrow^*$, then $n'$ is either $n$ or a left sibling of $n$.
11. if $p = p_1/p_2$, then there exists a node $v$ in $T$ such that $T \models p_1(n, v) \wedge p_2(v, n')$;
12. if $p = p_1 \cup p_2$, then $T \models p_1(n, n') \vee p_2(n, n')$;
13. if $p = p_1[q]$, then $T \models p_1(n, n')$ and $T \models q(n')$, where $q$ is a unary predicate of the following cases:
    (a) $q$ is $p_2$: then $T \models \exists n''\ p_2(n', n'')$;
    (b) $q$ is $lab() = A$: then the label of $n'$ is $A$;
    (c) $q$ is $p_2/@a\ \mathsf{op}\ `c'$: then $T \models \exists n_1\ (p_2(n', n_1)\ \wedge\ n_1.a\ \mathsf{op}\ `c')$, where $n_1.a$ denotes the value of the $a$ attribute of $n_1$; that is, there exists a node $n_1$ in $T$ such that $T \models p_2(n', n_1)$, $n_1$ has attribute $a$ and $n_1.a\ \mathsf{op}\ `c'$;
    (d) $q$ is $p_2/@a\ \mathsf{op}\ p_2'/@b$: then $T$ satisfies the existential formula: $T \models \exists n_1\ \exists n_2\ (p_2(n', n_1) \wedge\ p_2'(n', n_2)\ \wedge\ n_1.a\ \mathsf{op}\ n_2.b)$;
    (e) $q$ is $q_1 \wedge q_2$: then $T \models (q_1(n')\ \wedge\ q_2(n'))$;
    (f) $q$ is $q_1 \vee q_2$: then $T \models (q_1(n')\ \vee\ q_2(n'))$;
    (g) $q$ is $\neg q'$: then $T \not\models q'(n')$; for instance, if $q$ is $\neg p_2$, then $T \models \forall n''\ \neg p_2(n', n'')$.

Here $n$ is referred to as the *context node*. If $T \models p(n, n')$ then we say that $n'$ is *reachable* from $n$ via $p$. We use $n[\![p]\!]$ to denote the set of all the nodes reached from $n$ via $p$, i.e., $n[\![p]\!] = \{n' \mid n' \in T,\ T \models p(n, n')\}$.

We investigate various fragments of $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \leftarrow, \leftarrow^*, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg)$. We denote a fragment $\mathcal{X}$ by listing the operators supported by $\mathcal{X}$: the presence or absence of negation '$\neg$', data-value joins '$=, \neq$', upward traversal '$\uparrow$' ('$\uparrow^*$'), sideways traversal '$\leftarrow$' ('$\leftarrow^*$') and '$\rightarrow$' ('$\rightarrow^*$'), wildcard '$\downarrow$', recursive axis '$\downarrow^*$, $\uparrow^*$, $\leftarrow^*$', and '$\rightarrow^*$', qualifiers '$[\,]$', and union and disjunction '$\cup$'. The concatenation operator '$/$' is *included in all fragments by default.*

*Example 2.* Consider the XML tree $T$ of $D_2$ shown in Fig. 1, and the following XPath queries. (a) Over $T$, $\downarrow^*[\downarrow/\rightarrow[\mathsf{lab}() = X]]$ is to find all the nodes in $T$ that have child whose right sibling is labeled $X$. This query is in the fragment $\mathcal{X}(\downarrow, \downarrow^*, \rightarrow, [\,])$. (b) Posed on $T$, $\downarrow^*[\neg\downarrow^*[X/\rightarrow[\mathsf{lab}() = Y]]]$ is to find all the nodes in $T$ that have no descendant which has children $X$ and $Y$ in this order. This query is in $\mathcal{X}(\downarrow, \downarrow^*, \rightarrow, [\,], \neg)$. (c) Over the XML tree $T_1$ of $D_1$ shown in Fig. 1, $\downarrow^*[A/\rightarrow/\rightarrow^*[\mathsf{lab}() = A] \wedge \neg(B/\rightarrow/\rightarrow^*[\mathsf{lab}() = B]/\rightarrow/\rightarrow^*[\mathsf{lab}() = B])]$ is to find all the nodes that have at least two $A$ children but at most three $B$ children. It is in $\mathcal{X}(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, [\,], \neg)$.

### 2.3 The Satisfiability Problem

We say that an XML tree $T$ *satisfies* a query $p$, denoted by $T \models p$, iff $T \models \exists n\ p(r, n)$, where $r$ is the root of $T$. In other words, $r[\![p]\!] \neq \emptyset$. We focus on the satisfiability of XPath queries applied to the root of $T$. The complexity results of this paper remain intact for arbitrary context nodes.

We study the satisfiability problem for XPath queries considered together with a DTD. That is the problem to determine whether a given XPath query $p$ and a DTD $D$ are satisfiable by an XML tree. We say that an XML tree $T$ *satisfies* $p$ *and* $D$, denoted by $T \models (p, D)$, if $T \models p$ and $T \models D$. If such a $T$ exists, we say that $(p, D)$ is *satisfiable*.

Formally, for a fragment $\mathcal{X}$ of XPath we define the *XPath satisfiability problem* $\mathsf{SAT}(\mathcal{X})$ as follows:

| | |
|---|---|
| PROBLEM: | $\mathsf{SAT}(\mathcal{X})$ |
| INPUT: | A DTD $D$, an XPath query $p$ in $\mathcal{X}$. |
| QUESTION: | Is there an XML document $T$ such that $T \models (p, D)$? |

We are also interested in the complexity of the satisfiability analysis in the query size alone. The *satisfiability problem for a fragment* $\mathcal{X}$ *in the absence of DTDs* is the problem of determining, given any query $p$ in $\mathcal{X}$, whether or not there is an XML tree $T$ such that $T \models p$. As shown in [2], this problem is a *special case* of $\mathsf{SAT}(\mathcal{X})$, when DTDs $D$ are restricted to have a certain syntactic form. Since such DTDs can be computed in low polynomial of the size of the input queries, all the lower bounds for $\mathsf{SAT}(\mathcal{X})$ established in this paper, except Proposition 6, also hold in the absence of DTDs.

## 3 Horizontal Versus Vertical Traversal

In this section we study the basic properties of XPath fragments with sibling axes, and explore the connection between these fragments and the corresponding fragments without sibling axes.

**Increase in Expressive Power.** We first show that the sibling axes do add expressive power to fragments without horizontal modalities.

**Proposition 1.** *The sibling axes are not expressible in* $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg)$*, our largest fragment with only vertical axes.* ∎

*Proof.* Consider an XPath query $Q = A/{\rightarrow}$, and two XML trees $T_1$ and $T_2$, where $T_1$ consists of a root with two $A$ children, and $T_2$ has a root with three $A$ children. Over $T_1$ and $T_2$, $Q$ is to find all $A$ children of the root except the very first one. One can verify that $Q$ is not expressible in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg)$, in which $T_1$ and $T_2$ are not distinguishable. Similarly for $\leftarrow, \rightarrow^*$ and $\leftarrow^*$. ∎

We say that an XPath fragment $\mathcal{X}$ has the *finite model property* if for any query $p$ in $\mathcal{X}$, if $p$ is satisfiable by a (possibly infinite) tree, then there exists a finite tree that satisfies $p$. An XPath fragment $\mathcal{X}$ has the *small model property* if there exists a recursive function $f$ such that for each $p \in \mathcal{X}$, if $p$ is satisfiable, then $p$ has a finite model of size at most $f(|p|)$, where $|p|$ is the size of $p$.

As another evidence for the increase of expressive power, observe that the fragment $\mathcal{X}(\rightarrow, [\,], \neg)$ does not have the finite model property. Indeed, the query $\epsilon[A \wedge \neg A[\neg{\rightarrow}[\mathsf{lab}() = A]]]$ does not have the finite model. Thus we have:

**Proposition 2.** *The satisfiability problem for any fragment that subsumes* $\mathcal{X}(\rightarrow, [\,], \neg)$ *does not have the finite model property, in the presence of DTDs and in the absence of DTDs.* ∎

In contrast, [2] has shown the following: (a) $\mathcal{X}(\downarrow, \uparrow, \cup, [\,], \neg)$ has the small model property in the presence of DTDs and in the absence of DTDs, and (b) $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], \neg)$ has the small model property over non-recursive DTDs. This shows that the sibling axes may complicate the satisfiability analysis.

**DTD Coding.** We next show that certain DTDs can be encoded in terms of a qualifier in $\mathcal{X}(\downarrow, \downarrow^*, \rightarrow, [\,], \neg)$. Recall the following from [2]: a *normalized DTD* restricts its productions $A \rightarrow \alpha$ such that $\alpha$ is of the following forms:

$$\alpha \quad ::= \quad \epsilon \quad | \quad B_1, \ldots, B_n \quad | \quad B_1 + \cdots + B_n \quad | \quad B^*$$

where $B_i$ is a type in $Ele$. It was shown there that any DTD can be "normalized" in linear time, and moreover, for any XPath fragment with $\cup$ and $\downarrow$ and without sibling axes, the normalization has no impact on the complexity bounds of its satisfiability analysis. Below we further show that we can actually encode a normalized DTD in terms of XPath qualifiers in $\mathcal{X}(\downarrow, \downarrow^*, \rightarrow, [\,], \neg)$.

**Proposition 3.** *A normalized DTD* $D$ *can be expressed as a qualifier* $q_D$ *in any XPath fragment that subsumes* $\mathcal{X}(\downarrow, \downarrow^*, \rightarrow, [\,], \neg)$*. That is, for any query* $Q$ *in a fragment that subsumes* $\mathcal{X}(\downarrow, \downarrow^*, \rightarrow, [\,], \neg)$*,* $(Q, D)$ *is satisfiable iff* $\epsilon[q_D]/Q$ *is satisfiable in the absence of DTDs.* ∎

*Proof.* We show that for any $A$ in the set $Ele$ of the element types of a normalized DTD, the production $A \rightarrow P(A)$ can expressed as a qualifier $Q^A$ in $\mathcal{X}(\downarrow, \downarrow^*, \rightarrow, [\,], \neg)$, by induction on the structure of $P(A)$. Putting these together, we obtain a single qualifier $q_D = \epsilon[\bigwedge_{A \in Ele} Q^A]$ at the root. ∎

As an immediate result, for any XPath fragment $\mathcal{X}(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, [\,], \neg, \ldots)$, its satisfiability analysis in the presence of normalized DTDs is equivalent to its counterpart in the absence of DTDs.

In contrast, below we show that normalized DTDs are not expressible in fragments without sibling axes. Indeed, it was shown in [2] that without sibling axes, the lower bounds for XPath satisfiability analysis in the presence of DTDs typically do not carry over to the counterpart in the absence of DTDs, although the analysis without DTDs is a special case of its counterpart with DTDs.

**Proposition 4.** *A normalized DTD $D$ cannot be expressed as a qualifier $q_D$ in* $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg)$. ∎

*Proof.* One can verify that two different DTDs $D_1$ and $D_2$ are not distinguishable by any XPath query in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg)$, where $D_1$ has a single production $r \rightarrow A, A$, and $D_2$ consists of a single production $r \rightarrow A, A, A$. ∎

**Encoding Horizontal Traversal in Terms of Vertical Modalities**. Let $\mathcal{X}(\rightarrow, \rightarrow^*, [\,], \ldots)$ be any class of XPath queries that allows '$\rightarrow, \rightarrow^*$' and qualifiers. Let $\mathcal{X}^*(\downarrow, \uparrow, \cup, [\,], \ldots)$ be a variation of $\mathcal{X}(\rightarrow, \rightarrow^*, [\,], \ldots)$ by (a) supporting $\downarrow, \uparrow$, and $\cup$, (b) supporting the general Kleene closure defined by $\beta^*$, where $\beta$ is a simple path $A_1[q_1]/\ldots/A_k[q_n]$, where $A_i$ is a label and $[q_i]$ is a Boolean combination of simple label testing qualifiers (of the form $\mathsf{lab}() = A$), and (c) discarding any queries with '$\rightarrow, \rightarrow^*$'. Note that $\mathcal{X}^*(\downarrow, \uparrow, \cup, [\,], \ldots)$ is far more restrictive than the regular XPath fragment introduced and studied in [17].

**Proposition 5.** *For any class $\mathcal{X}(\rightarrow, \rightarrow^*, [\,], \ldots)$ of XPath queries, there exists a PTIME computable function $N$ from DTDs to DTDs, and there exists a PTIME computable function $f$ from queries in $\mathcal{X}(\rightarrow, \rightarrow^*, [\,], \ldots)$ to queries in $\mathcal{X}^*(\downarrow, \uparrow, \cup, [\,], \ldots)$ such that, for any DTD $D$ and any XPath query $p \in \mathcal{X}(\rightarrow, \rightarrow^*, [\,], \ldots)$, there exists an XML tree $T$ such that $T \models (p, D)$ iff there exists an XML tree $T'$ such that $T' \models (f(p), N(D))$.* ∎

*Proof.* The mapping $N$ is based on the canonical binary encoding of instances of the input $D$, which introduces new labels. Then $f$ can be defined such that it traverses "descendants" and "siblings" by visiting left subtrees and right subtrees in the binary trees, respectively. The query translation requires the use of $\downarrow, \uparrow, \cup, [\,]$ and simple paths of the form $A_1[q_1]/\ldots/A_k[q_n]$ as described above. ∎

This tells us that, upon the availability of upper bounds for conditional and regular XPath fragments [17] without siblings, the bounds can carry over to our fragments with sibling axes.

## 4 Complexity of XPath Satisfiability with Sibling Axes

In this section we study the satisfiability problem for various XPath fragments with sibling axes, and contrast the complexity bounds with their counterparts

for the corresponding fragments without sibling axes. To understand the impact of different XPath modalities on the satisfiability analysis, we start with a simple fragment $\mathcal{X}(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \cup)$, and then extend the fragment gradually by adding qualifiers, data-value joins, and negation one by one. To study the interaction between XPath modalities and DTD constructs, we also consider the analysis under DTDs restricted to have certain constructs and in the absence of DTDs.

### 4.1 XPath Fragments Without Qualifiers

Without sibling axes, the absence of qualifiers simplifies the satisfiability analysis [2]. Below we show that it is also the case for XPath fragments with siblings.

**Proposition 6.** $\mathsf{SAT}(\mathcal{X}(\downarrow^*))$ *is NLOGSPACE-hard in the presence of DTDs.* ■

*Proof.* This can be verified by LOGSPACE reduction from directed graph connectivity with specified source and target, which is NLOGSPACE-hard [24]. ■

In the absence of DTDs, all queries in $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ are always satisfiable [2].

**Theorem 1.** *Both* $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \cup))$ *and* $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \leftarrow, \leftarrow^*, \cup))$ *are NLOGSPACE-complete in the presence of DTDs.*

*Proof.* We provide a NLOGSPACE algorithm for checking the satisfiability of $(Q, D)$ for an input DTD $D$ and query $Q \in \mathcal{X}(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \cup)$ (resp. $\leftarrow, \leftarrow^*$). The key idea is to code vertical navigation using a query graph $G_Q$ of $Q$ and horizontal moves using NFAs of the regular expressions in $D$. This only requires us to store triplets $(q, v, A)$ at each step, where $q$ is a NFA state, $v$ is node in $G_Q$ and $A$ is a label. This only needs LOGSPACE. ■

Recall that $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup))$ is in PTIME [2], which contains NLOGSPACE. Thus Theorem 1 tells us that in the absence of qualifiers, the addition of sibling axes does not complicate the satisfiability analysis. As another evidence:

**Theorem 2.** $\mathsf{SAT}(\mathcal{X}(\rightarrow, \leftarrow))$ *is in PTIME in the presence of DTDs.* ■

In contrast, $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$ is NP-hard [2]. The difference between $\mathcal{X}(\downarrow, \uparrow)$ and $\mathcal{X}(\rightarrow, \leftarrow)$ is that while a query in $\mathcal{X}(\downarrow, \uparrow)$ can constrain the subtree of a node by moving downward and upward repeatedly in the subtree, queries in $\mathcal{X}(\rightarrow, \leftarrow)$ are not able to do it: as soon as the navigation moves down in a tree, it cannot move back to the same node. Leveraging this we are able to develop a PTIME algorithm, based on dynamic programming, for deciding the satisfiability of $(Q, D)$ for a given DTD $D$ and query $Q \in \mathcal{X}(\rightarrow, \leftarrow)$.

From these we can see that XPath queries with sibling axes are quite well behaved in the absence of qualifiers.

### 4.2 Positive XPath Queries with Qualifiers

We now consider *positive* XPath fragments, i.e., fragments supporting qualifiers but not including negation ($\neg$). Positive fragments are contained in positive existential two-variable first-order logic over trees, with binary predicates *child*, *descendant*, and *sibling* [17]. It is known that qualifiers make the satisfiability analysis harder for XPath fragments without siblings [2]. We show that this is also the case when sibling axes are considered instead of vertical modalities.

**Theorem 3.** *The satisfiability problem for the following fragments is NP-hard:*

1. $\mathsf{SAT}(\mathcal{X}([\,]))$ *under nonrecursive DTDs;*
2. $\mathsf{SAT}(\mathcal{X}(\rightarrow,[\,]))$ *and* $\mathsf{SAT}(\mathcal{X}(\leftarrow,[\,]))$ *under fixed, disjunctive-free and nonrecursive DTDs;*
3. $\mathsf{SAT}(\mathcal{X}(\rightarrow,\cup,[\,]))$ *and* $\mathsf{SAT}(\mathcal{X}(\leftarrow,\cup,[\,]))$ *in the absence of DTDs.* ∎

*Proof.* These can be verified by reduction from the 3SAT problem, which is NP-complete (cf. [24]). ∎

Here by *fixed DTDs* we mean that the input to the satisfiability analysis consists of only a query rather than both a query and a DTD, and the XML trees considered are required to conform to a predefined DTD.

Contrast these with the following results in [2]. (a) $\mathsf{SAT}(\mathcal{X}(\downarrow,[\,]))$ is NP-hard under normalized DTDs. Here we improve that result by showing that $\mathsf{SAT}(\mathcal{X}([\,]))$ is already intractable under (not necessarily normalized) DTDs. (b) While $\mathsf{SAT}(\mathcal{X}(\downarrow,[\,]))$ is NP-complete for arbitrary DTDs, but it is in PTIME when DTDs are restricted to be disjunction-free. In contrast, Theorem 3 shows that it is no longer the case when $\downarrow$ is replaced by $\rightarrow$ or $\leftarrow$. (c) In the absence of DTDs, $\mathsf{SAT}(\mathcal{X}(\downarrow,\cup,[\,]))$ is in PTIME, as opposed to Theorem 3. Thus sibling axes complicate the satisfiability analysis in the presence of qualifiers.

Recall that $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,\uparrow,\uparrow^*,\cup,[\,],=))$ is in NP [2]. The result below shows that the addition of the sibling axes does not increase the upper bound.

**Theorem 4.** $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,\uparrow,\uparrow^*,\leftarrow,\leftarrow^*,\rightarrow,\rightarrow^*,\cup,[\,],=))$ *is in NP.* ∎

*Proof.* It suffices to show that $\mathsf{SAT}(\mathcal{X}^*(\downarrow,\downarrow^*,\uparrow,\uparrow^*,\cup,[\,],=))$ is in NP by Proposition 5. A NP decision algorithm is then provided for this fragment, by extending the NP algorithm for $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,\uparrow,\uparrow^*,\cup,[\,],=))$ developed in [2]. ∎

### 4.3 XPath Fragments with Negation

In contrast to positive XPath fragments, negation introduces universal quantifiers and complicates the satisfiability analysis without sibling axes [2]. We show that in the presence of sibling axes the situation is also bad, and may be worse.

It is known that $\mathsf{SAT}(\mathcal{X}(\downarrow,[\,],\neg))$ is PSPACE-hard in the presence of DTDs [2]. We show that the lower bound remains intact if we substitute $\rightarrow$ (resp. $\leftarrow$) for $\downarrow$ in the fragment, even when the DTDs are restricted or left out.

**Theorem 5.** $\mathsf{SAT}(\mathcal{X}(\rightarrow, [\,], \neg))$ *and* $\mathsf{SAT}(\mathcal{X}(\leftarrow, [\,], \neg))$ *are PSPACE-hard in the following settings: (1) under non-recursive and no-star DTDs; and (2) in the absence of DTDs.* ∎

*Proof.* The lower bounds can be proved by reduction from 3QSAT, a well-known PSPACE-complete problem (cf. [24]). ∎

**Theorem 6.** $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \leftarrow, \leftarrow^*, \rightarrow, \rightarrow^*, \cup, [\,], \neg))$ *is PSPACE-complete under no-star DTDs.* ∎

*Proof.* The upper bound can be verified by reduction to $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$, based on a variation of the proof of Proposition 5. ∎

It is known [17] that $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,], \neg))$ is EXPTIME-hard and that $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \leftarrow, \leftarrow^*, \rightarrow, \rightarrow^*, \cup, [\,], \neg))$ is in EXPTIME. We now show that we already have the EXPTIME hardness in the presence of *neither recursion in XPath nor recursion in DTDs.*

**Theorem 7.** $\mathsf{SAT}(\mathcal{X}(\uparrow, \rightarrow, [\,], \neg))$ *is EXPTIME-hard under fixed, nonrecursive and disjunction-free DTDs.* ∎

This can be verified by reduction from the two-player game of corridor tiling, which is EXPTIME-complete (cf. [4]). To see why the result holds, observe the following. One can encode a certain recursive DTD $D_1$ in terms of a "flattened" DTD $D_2$, and based on this a mapping $N$ can be defined from XML trees of $D_1$ to XML trees of $D_2$ via "unnesting"; furthermore, there is a mapping $f$ such that for certain queries $Q$ in $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,], \neg)$, $f(Q)$ is in $\mathcal{X}(\uparrow, \rightarrow, [\,], \neg)$ and moreover, if $Q$ is satisfiable by an XML tree $T$ of $D_1$, then $f(Q)$ is satisfiable by $N(T)$. In $N(T)$, the child, parent and right sibling axes suffice to access certain elements that are deep in $T$. From this it follows that a reduction from the two-player game of corridor tiling to $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,], \neg))$ can be coded in terms of a query in $\mathcal{X}(\uparrow, \rightarrow, [\,], \neg)$ and a fixed, nonrecursive DTD as described above. This explains why the EXPTIME lower bounds is robust in the absence of XPath and DTD recursions, and demonstrates the power of sibling axes.

### 4.4 XPath Fragments with Negation and Data Values

Finally, we investigate the satisfiability analysis for XPath fragments with data-value joins, negation and sibling axes. As observed in [2], the interaction between data-value joins and negation is already intricate in the absence of sibling axes. Indeed, $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg))$ is undecidable in presence of fixed recursive DTDs [2]. However, it is not yet known whether or not the undecidability result still holds (a) under non-recursive DTDs, (b) under fixed DTDs, and (c) in the absence of DTDs. In contrast, we next show that in the presence of sibling axes but without vertical XPath recursion $\downarrow^*$ and $\uparrow^*$, the problem remains undecidable in all the settings mentioned above.

**Theorem 8.** $\mathsf{SAT}(\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg))$ *is undecidable in any of the following setting: (1) under non-recursive, fixed and disjunction-free DTDs; and (2) in the absence of DTDs.* ∎

The undecidability result can be verified by reduction from the halting problem for two-register machines, which is known to be undecidable (see, e.g., [4]). The proof extends the undecidability proof of [2] for $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, [\,], =, \neg)$ under fixed recursive DTDs, by "flattening" DTDs in the same way as mentioned above. The proof leverages the following observation: by means of XPath qualifiers with $\rightarrow, \rightarrow^*, \leftarrow$ and $\uparrow$, (a) DTD linear recursion introduced by productions of the form $A \rightarrow A + \epsilon$ can be coded with productions of the form $A \rightarrow B^*$; (b) disjunction in a DTD can also be coded in terms of the use of Kleene star. This allows us to get rid of linear recursion and disjunction required by the undecidability proof of [2], and again shows the expressive power of sibling axes.

## 5 The Containment Analysis for XPath with Siblings

In this section we present a few lower bounds for the containment analysis of XPath fragments with sibling axes, by exploring the connection between the containment analysis and its satisfiability counterpart, and by using the complexity results for the satisfiability analysis given in the last section.

The *containment problem* for a fragment $\mathcal{X}$ in the presence of DTDs, denoted by $\mathsf{CNT}(\mathcal{X})$, is the problem to determine, given any queries $Q_1, Q_2 \in \mathcal{X}$ and a DTD $D$, whether or not for any XML tree $T$ of $D$, $r[\![Q_1]\!] \subseteq r[\![Q_2]\!]$, where $r$ is the root of $T$. If this holds then we say that $Q_1 \subseteq Q_2$ *under* $D$.

It is easy to see that for any fragment $\mathcal{X}$, $\mathsf{SAT}(X)$ is reducible to the complement of $\mathsf{CNT}(\mathcal{X})$. Recall that for a complexity class $\mathsf{K}$, co$\mathsf{K}$ stands for $\{\bar{P} \mid P \in \mathsf{K}\}$.

**Proposition 7.** *[2] For any class $\mathcal{X}$ of XPath queries, if $\mathsf{CNT}(\mathcal{X})$ is in $\mathsf{K}$ for some complexity class $\mathsf{K}$, then $\mathsf{SAT}(\mathcal{X})$ is in co$\mathsf{K}$. Conversely, if $\mathsf{SAT}(\mathcal{X})$ is $\mathsf{K}$-hard, then $\mathsf{CNT}(\mathcal{X})$ is co$\mathsf{K}$-hard.*

From this and Theorems 3, 5, 7 and 8 it immediately follows:

**Corollary 1.** *For the containment problem,*

1. $\mathsf{CNT}(\mathcal{X}(\rightarrow, [\,]))$ *and* $\mathsf{CNT}(\mathcal{X}(\leftarrow, [\,]))$ *are coNP-hard under fixed, disjunction-free and nonrecursive DTDs;*
2. $\mathsf{CNT}(\mathcal{X}(\rightarrow, \cup, [\,]))$ *and* $\mathsf{CNT}(\mathcal{X}(\leftarrow, \cup, [\,]))$ *are coNP-hard in the absence of DTDs;*
3. $\mathsf{CNT}(\mathcal{X}(\rightarrow, [\,], \neg))$ *and* $\mathsf{CNT}(\mathcal{X}(\leftarrow, [\,], \neg))$ *are PSPACE-hard (a) under nonrecursive and no-star DTDs, and (b) in the absence of DTDs;*
4. $\mathsf{CNT}(\mathcal{X}(\uparrow, \rightarrow, [\,], \neg))$ *is EXPTIME-hard under fixed, disjunction-free and nonrecursive DTDs;*
5. $\mathsf{CNT}(\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg))$ *is undecidable (a) under non-recursive, disjunction-free and fixed DTDs, and (b) in the absence of DTDs.* ∎

These are among the first lower bound results for the containment problem for XPath fragments with sibling axes. Indeed, the only other result that we are aware of is the EXPTIME lower bound given by [17] for $\mathsf{CNT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,], \neg))$. Corollary 1 strengthens that result by showing that $\mathsf{CNT}(\mathcal{X}(\uparrow, \rightarrow, [\,], \neg))$ is already EXPTIME-hard under restricted DTDs.

As observed in [2], the upper bound for $\mathsf{SAT}(\mathcal{X})$ is often much lower than its counterpart for $\mathsf{CNT}(\mathcal{X})$. However, for certain fragments $\mathcal{X}$ without sibling axes, $\mathsf{SAT}(\mathcal{X})$ and $\mathsf{CNT}(\mathcal{X})$ actually coincide. These include the following: (a) the class $\mathcal{X}_{(bl,\ [\,],\neg)}$ of *Boolean queries*, i.e., queries of the form $\epsilon[q]$, in any class $\mathcal{X}(\ldots, [\,], \neg)$ with negation and qualifiers; and (b) any class containing negation and closed under the *inverse operator* that is defined as a simple extension of $\mathsf{inverse}(\downarrow) = \uparrow$, $\mathsf{inverse}(\downarrow^*) = \uparrow^*$, $\mathsf{inverse}(\uparrow) = \downarrow$ and $\mathsf{inverse}(\uparrow^*) = \downarrow^*$.

We next show that this result of [2] carries over to XPath fragments with sibling axes, by extending (a) the class $\mathcal{X}_{(bl,\ [\,],\neg)}$ by including Boolean queries with sibling axes; (b) the definition of $\mathsf{inverse}$ such that $\mathsf{inverse}(\leftarrow) = \rightarrow$, $\mathsf{inverse}(\leftarrow^*) = \rightarrow^*$, $\mathsf{inverse}(\rightarrow) = \leftarrow$, $\mathsf{inverse}(\rightarrow^*) = \leftarrow^*$.

**Proposition 8.** *For any class $\mathcal{X}_{(bl,\ [\,],\neg)}$ of Boolean queries, $\mathsf{CNT}(\mathcal{X}_{(bl,\ [\,],\neg)})$ is reducible in constant time to the complement of $\mathsf{SAT}(\mathcal{X}_{(bl,\ [\,],\neg)})$. For any class $\mathcal{X}$ with negation and closed under $\mathsf{inverse}$, $\mathsf{CNT}(\mathcal{X})$ is reducible in linear time to the complement of $\mathsf{SAT}(\mathcal{X})$.* ∎

## 6 Conclusions

We have established complexity bounds for a number of XPath fragments with sibling axes, in the presence of DTDs, in the absence of DTDs, and under various restricted DTDs. The main results of the paper are summarized in Table 1. As immediate corollaries of these results, we have also provided several lower bounds

**Table 1.** The complexity of $\mathsf{SAT}(\mathcal{X})$ for various fragments $\mathcal{X}$ under different DTDs

| | | |
|---|---|---|
| NLOGSPACE -comp. | $\mathcal{X}(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \cup)$, $\mathcal{X}(\downarrow, \downarrow^*, \leftarrow, \leftarrow^*, \cup)$ | any DTDs<br>nonrec. DTDs |
| PTIME | $\mathcal{X}(\leftarrow, \rightarrow)$ | any DTD |
| NP-hard | $\mathcal{X}([\,])$ | nonrec DTDs |
| NP-hard | $\mathcal{X}(\leftarrow, [\,])$, $\mathcal{X}(\rightarrow, [\,])$ | fixed, '+'-free, nonrec DTDs |
| NP-hard | $\mathcal{X}(\leftarrow, \cup, [\,])$, $\mathcal{X}(\rightarrow, \cup, [\,])$ | no DTDs |
| NP-comp. | $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \leftarrow, \leftarrow^*, \rightarrow, \rightarrow^*, \cup, [\,], =)$ | any DTD |
| PSPACE-hard | $\mathcal{X}(\rightarrow, [\,], \neg)$, $\mathcal{X}(\leftarrow, [\,], \neg)$ | nonrec, no-star DTDs<br>no DTDs |
| PSPACE-comp. | $\mathcal{X}(\downarrow, \uparrow, \leftarrow, \leftarrow^*, \rightarrow, \rightarrow^*, \cup, [\,], \neg)$ | no-star DTDs |
| EXPTIME-hard | $\mathcal{X}(\uparrow, \leftarrow, [\,], \neg)$ | fixed, '+'-free, nonrec. DTDs |
| undecidable | $\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], \neg, =)$ | fixed, '+'-free, nonrec DTDs<br>no DTDs |

for the containment problem for XPath queries. Our main conclusion is that while sibling axes do not complicate the satisfiability analysis in the absence of qualifiers, they do make our lives harder in the presence of qualifiers.

To the best of our knowledge, the results of this paper are among the first results for the satisfiability and containment analyses of XPath fragments with sibling axes. They are complementary to the recent study on the satisfiability problem for XPath fragments without sibling axes [2]. They are useful not only for XML query and update optimization, but also for the static analysis of inference control for XML security, among other things.

There is naturally much more to be done. One open problem is to close the complexity gaps. For example, we do not know yet whether $\mathsf{SAT}(\mathcal{X}([\,]))$ is still intractable under fixed and disjunction-free DTDs, and whether or not $\mathsf{SAT}(\mathcal{X}(\rightarrow, [\,], \neg))$ is in PSPACE under arbitrary DTDs. Another topic for future work is to study the satisfiability problem for XPath in the presence of XML Schema, which typically consists of both a type (a specialized DTD) and a set of XML constraints. This setting was considered in [8] for the containment analysis.

**Acknowledgment.** The authors would like to thank Frank Neven for giving the proof idea for Theorem 1. Wenfei Fan is supported in part by EPSRC GR/S63205/01, EPSRC GR/T27433/01 and NSFC 60228006. Floris Geerts is postdoctoral researcher of the FWO Vlaanderen and is supported in part by EPSRC GR/S63205/01.

## References

1. S. Amer-Yahia, S. Cho, L. Lakshmanan, and D. Srivistava. Minimization of tree pattern queries. In *SIGMOD*, 2001.
2. M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. In *PODS*, 2005.
3. M. Benedikt, W. Fan, and G. M. Kuper. Structural properties of XPath fragments. In *ICDT*, 2003.
4. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 1997.
5. T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation, Feb 1998. `http://www.w3.org/TR/REC-xml`.
6. P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. *Computer Networks*, 39(5):473–487, 2002.
7. J. Clark and S. DeRose. *XML Path Language (XPath)*. W3C Recommendation, Nov. 1999.
8. A. Deutsch and V. Tannen. Containment for classes of XPath expressions under integrity constraints. In *KRDB*, 2001.
9. A. Deutsch and V. Tannen. Reformulation of XML queries and constraints. In *ICDT*, 2003.
10. W. Fan, C. Chan, and M. Garofalakis. Secure XML querying with security views. In *SIGMOD*, 2004.
11. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *VLDB*, 2002.

12. G. Gottlob, C. Koch, and K. Schulz. Conjunctive queries over trees. In *PODS*, 2004.
13. J. Hidders. Satisfiability of XPath expressions. In *DBPL*, 2003.
14. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation (2nd Edition)*. Addison Wesley, 2000.
15. L. Lakshmanan, G. Ramesh, H. Wang, and Z. Zhao. On testing satisfiability of tree pattern queries. In *VLDB*, 2004.
16. L. Libkin. Logics over unranked trees: an overview. In *ICALP*, 2005.
17. M. Marx. XPath with conditional axis relations. In *EDBT*, 2004.
18. M. Marx. First order paths in ordered trees. In *ICDT*, pages 114–128, 2005.
19. G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *JACM*, 51(1):2–45, 2004.
20. M. Murata. Extended path expressions for XML. In *PODS*, 2001.
21. F. Neven and T. Schwentick. Expressive and efficient languages for tree-structured data. In *PODS*, 2000.
22. F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *ICDT*, 2003.
23. D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking forward. In *XMLDM*, 2002.
24. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
25. T. Schwentick. Xpath query containment. *SIGMOD Rec.*, 33(1):101–109, 2004.
26. G. Sur, J. Hammer, and J. Siméon. An XQuery-based language for processing updates in XML. In *PLAN-X*, 2004.
27. H. Thompson et al. XML Schema. W3C Recommendation, Oct. 2004. `http://www.w3.org/TR/xmlschema1.`
28. P. T. Wood. Minimising simple XPath expressions. In *WebDB*, 2001.
29. P. T. Wood. Containment for XPath fragments under DTD constraints. In *ICDT*, 2003.