Satisfiability of XPath Queries with Sibling Axes

Floris Geerts[1] and Wenfei Fan[2]

[1] University of Limburg 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

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 de-

cidable 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, 16–18, 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 $A \to P(A)$ 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: $\quad r \to X^*, \qquad X \to (A, B^*)^*$
$Att = \emptyset, \quad R(X) = R(T) = R(F) = \emptyset$.

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

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

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

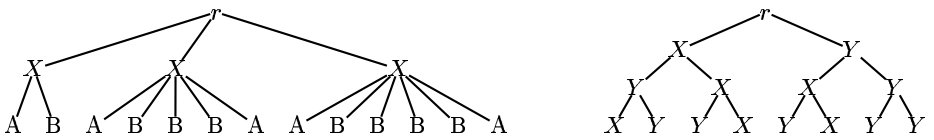

Fig. 1. XML trees of the DTDs D_1 (left) and D_2 (right) given in Example 1.

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 Σ 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 ϵ and A denote the empty path (the *self-axis*) and a label in Σ (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 Σ, 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:

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

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 $p_2/@a$ op 'c': then $T \models \exists n_1\ (p_2(n',n_1)\ \wedge\ n_1.a$ 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 $n_1.a$ op 'c';
    (d) q is $p_2/@a$ op $p_2'/@b$: then T satisfies the existential formula: $T \models \exists n_1\ \exists n_2\ (p_2(n',n_1) \wedge\ p_2'(n',n_2)\ \wedge\ n_1.a$ 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 α 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 $r \to A, A$, and D_2 consists of a single production $r \to A, A, A$. ∎

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 β^*, where β 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 con-

tainment 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 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 K for some complexity class K, then $\mathsf{SAT}(\mathcal{X})$ is in co$\mathsf{K}$. Conversely, if $\mathsf{SAT}(\mathcal{X})$ is 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 non-recursive 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 $\mathsf{inverse}(\leftarrow) = \rightarrow$, $\mathsf{inverse}(\leftarrow^*) = \rightarrow^*$, $\mathsf{inverse}(\rightarrow) = \leftarrow$, $\mathsf{inverse}(\rightarrow^*) = \leftarrow^*$.

NLOGSPACE -comp.	$\mathcal{X}(\downarrow,\downarrow^*,\rightarrow,\rightarrow^*,\cup)$, $\mathcal{X}(\downarrow,\downarrow^*,\leftarrow,\leftarrow^*,\cup)$	any DTDs 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 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 no DTDs

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

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* 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 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.

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.

Appendix

Proofs

Proof of Proposition 1

Consider two XML trees T_1 and T_2, where T_1 consists of a root r with two A children, and T_2 consists of a root r_2 with three A children. It is known [17] that $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg)$ is contained in two-variable first-order logic with binary relations *child* and *descendants*, denoting parent-child and ancestor-descendant relations on trees, respectively. Using 2-pebble Ehrenfeucht-Fraïssé (EF) style game, one can show that T_1 and T_2 are equivalent in the two-variable first-order logic described above. In contrast, T_1 and T_2 are distinguishable by XPath query $Q = A/\rightarrow$, which yields different answers when posed against T_1 and T_2. Thus Q is not expressible in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg)$. ∎

Proof of Proposition 3

Let $D = (Ele, Att, P, R, r)$ be a normalized DTD, let $A \in Ele$ and T be an XML tree such that $T \models D$. We enforce the structure on T specified by D in terms of XPath qualifiers, based on the structure of D as follows:

(1) $P(A) = \epsilon$: We express that there exists no A node (labeled A) that has any children:

$$Q_\epsilon^A = \neg\downarrow^*[\mathsf{lab}() = A \wedge \downarrow].$$

(2) $P(A) = B_1, \ldots, B_n$: We express that there exists no A node that has a different (ordered) set of children than $B_1, \ldots, B_n$. First, we express that an A-labeled node has exactly n children:

$$Q_n^A = \neg\downarrow^*[\mathsf{lab}() = A \wedge \downarrow/\rightarrow^n \wedge \neg\downarrow/\rightarrow^{n+1})].$$

where $\rightarrow^n$ is a shorthand of the repetition of $\rightarrow$ for n times.

Next, we express that all $B_1, \ldots, B_n$ must appear under A in the right order:

$$Q_o^A = \neg\downarrow^*[\mathsf{lab}() = A \wedge \neg(\downarrow/\rightarrow[\mathsf{lab}() = B_1]/\ldots/\rightarrow[\mathsf{lab}() = B_n])],$$

The final qualifier then becomes $Q_c^A = Q_n^A \wedge \wedge Q_o^A$.

(3) $P(A) = B_1 + \cdots + B_n$. We express that there exists no A-node that has more or less than one child, and that one of the B_is appears as the child of A-labeled

nodes. More specifically, Q_1^A expresses that there is exactly one child, where Q_n^A is the qualifier of the previous case with $n = 1$. We express that one of the B_is appears as a child as follows:

$$Q_\vee^A = \neg\downarrow^*[\mathsf{lab}() = A \wedge \downarrow[\bigwedge_{i\in[1,n]} (\mathsf{lab}() \neq B_i)]].$$

The final qualifier then becomes $Q_d^A = Q_1^A \wedge Q_\vee^A$.

(4) $P(A) = B^*$. We express that all children of A-labeled nodes should be labeled with B. More specifically,

$$Q_*^A = \neg\downarrow^*[\mathsf{lab}() = A \wedge \downarrow[\mathsf{lab}() \neq B]].$$

It is clear that if we combine all the above qualifiers for each element type in the DTD in a conjunction $Q = \epsilon[\bigwedge_{A\in Ele} Q^A]$, where Q^A is one of the four qualifiers obtained above depending on $P(A)$, then for any tree T, $T \models Q$ if and only if $T \models Q$. By composing Q with the input query p, we obtain that $T \models (p, D)$ if and only if $T \models \epsilon[Q]/p$, as desired. □

Proof of Proposition 4

Consider two DTDs D_1 and D_2, where D_1 has a single production $r \to A, A$, and D_2 consists of a single production $r \to A, A, A$. Similarly to the proof of Proposition 1, one can show that D_1 and D_2 are equivalent in two-variable first-order logic with binary relations *child* and *descendants*, by using 2-pebble EF game. Thus D_1 and D_2 are not distinguishable by any XPath query in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg)$, since the fragment is contained in the two-variable first-order logic. As a result, D_1 and D_2 are not expressible as qualifiers in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg)$. ■

Proof of Proposition 5

The proof consists of two steps: First, we show how to convert any DTD D into a binary DTD $N(D)$ (i.e., a DTD describing a binary tree) such that if there exists a tree T conforming to D, then there exists a (binary) tree T_b conforming to D_b. Secondly, we show how to convert any XPath query p in $\mathcal{X}(\rightarrow, \rightarrow^*, [\,], \ldots)$ into an expression $f(p)$ in $\mathcal{X}^*(\downarrow, \uparrow, \cup, [\,], \ldots)$ such that if $T \models (p, D)$ if and only if $T_b \models (f(p), N(D))$.

To transform the DTD $D = (Ele, Att, P, R, r)$ we use the standard encoding into a binary tree where the left child represents the leftmost child and the right child represents the next right sibling (See Figure 2). We will distinguish between left

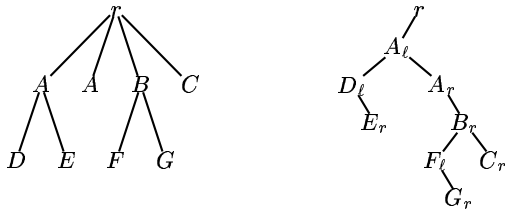

Fig. 2. Binary encoding of XML tree.

and right by introducing for each element type $A \in Ele$ a new element types A_ℓ (for left) and A_r (for right). The standard mapping of the DTD D into the DTD $N(D)$ describing the binary tree can be done in quadratic time.

We now show how to encode an XPath expression $p \in \mathcal{X}(\leftarrow, \leftarrow^*, [\,], \cup, \ldots)$ into an equivalent expression $f(p) \in \mathcal{X}^*(\downarrow, \uparrow, [\,], \cup, \ldots)$.

The encoding is by induction on the structure of p. We denote by Ele_ℓ (Ele_r) the union of all left (right) labeled element types in Ele'. We use the abbreviation $\epsilon[\mathsf{lab}() = Ele_\ell]$ for $\epsilon[\cup_{A \in Ele_\ell} \mathsf{lab}() = A]$ and similarly for label tests involving Ele_r.

We distinguish between the following cases:

- If $p = \epsilon$, then $f(p) = \epsilon$.
- If $p = A$, then $f(p) = A_\ell \cup \downarrow/(\downarrow[\mathsf{lab}() = Ele_r])^*/A_r$.
- If $p = \downarrow$, then $f(p) = \downarrow \cup \downarrow/(\downarrow[\mathsf{lab}() = Ele_r])^*$.
- If $p = \downarrow^*$, then $f(p) = \epsilon \cup \downarrow[\mathsf{lab}() = Ele_\ell]/\downarrow^*$.
- If $p = \uparrow$, then $f(p) = (\uparrow[\mathsf{lab}() = Ele_r])^*/\epsilon[\mathsf{lab}() = Ele_\ell]/\uparrow$.
- If $p = \uparrow^*$, then $f(p) = \uparrow^*[\mathsf{lab}() = Ele_\ell]/\uparrow$.
- If $p = \rightarrow$, then $f(p) = \downarrow[\mathsf{lab}() = Ele_r]$.
- If $p = \rightarrow^*$, then $f(p) = \epsilon \cup (\downarrow[\mathsf{lab}() = Ele_r])^*$.
- If $p = \leftarrow$, then $f(p) = \epsilon[\mathsf{lab}() = Ele_r]/\uparrow$.
- If $p = \leftarrow^*$, then $f(p) = \epsilon[\mathsf{lab}() = Ele_\ell] \cup \epsilon[\mathsf{lab}() = Ele_r]/(\uparrow[\mathsf{lab}() = Ele_r])^*$
- If $p = p_1 \cup p_2$, then $f(p) = f(p_1) \cup f(p_2)$.
- If $p = p_1/p_2$, then $f(p) = f(p_1)/f(p_2)$.
- If $p = \epsilon[q]$, then $f(\epsilon[q]) = \epsilon[f(q)]$.
  - If $q = p'$, then $f(q) = f(p')$.
  - If $q = \neg p'$, then $f(q) = \neg f(p')$.
  - if $q = \mathsf{lab}() = A$, then $f(q) =$ '$\mathsf{lab}() = A_\ell \cup \mathsf{lab}() = A_r$'.
  - If $q = p_1/@a$ op $p_2/@b$, then $f(q) = f(p_1)/@a$ op $f(p_2)/@b$.

We need to show that $T \models (p, D)$ if and only if $T_b \models (f(p), N(D))$ where T_b denote the binary encoding of T. By construction, it is clear that $T_b \models N(D)$. By induction on the structure of p and the semantics of XPath, it is easy to see that $T \models p$ if and only if $T_b \models f(p)$. ∎

Proof of Proposition 6

We show that $\mathsf{SAT}(\mathcal{X}(\downarrow^*))$ is NLOGSPACE-hard in the presence of DTDs by reduction from directed graph connectivity with specified source s and target t nodes. This problem is known to be NLOGSPACE-complete [24]. Given a graph $G = (V, E)$ and two designated source and target nodes s and t, we define the non-recursive and no-star DTD $D = (Ele, Att, P, R, r)$ as $Ele = V$, $Att = \emptyset$, $R = \emptyset$, $r = s$ and $P(v) = w_1, \ldots, w_k + \epsilon$ such that $(v, w_i) \in E$ for all $i \in [1, k]$. It is easy to see that $p = s/\downarrow^*/t$ is satisfiable by an XML tree $T \models D$ if and only if there exists a (directed) path in G from s to t. The reduction is clearly computable in LOGSPACE. ∎

Proof of Theorem 1

Let $D = (Ele, Att, P, R, r)$ be a DTD and $p \in \mathcal{X}(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \cup)$. We associate with each production $P(A)$ for $A \in Ele$ a finite automaton M_A corresponding to the regular expression $P(A)$. Furthermore, we assume that p is described by a query graph G_p defined constructively as follows: If $p = \epsilon, A, \downarrow, \downarrow^*, \rightarrow$ or $\rightarrow^*$, then G_p consists of a single node labeled with the p. This single node is both input and exit node. If $p = p_1/p_2$, then G_p is obtained by adding an edge between the exit node of G_{p_1} to the input node of G_{p_2} and by defining the input (exit) node of G_p to be the input (exit) node of G_{p_1} (G_{p_2}). Hence, in this case we combine the two graphs sequentially. Finally, if $p = p_1 \cup p_2$, then G_p is obtained by adding a common (dummy) input (exit) nodes of G_{p_1} and G_{p_2} that are labeled with ϵ. Hence, in this case we combine the two graphs in parallel. Note that we always have a single input s and exit node t in the query graph.

To show that $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \cup))$ is NLOGSPACE, we show that given the automata M_A for $A \in Ele$ and the query graph G_p on an input tape, we need to maintain only two pointers (ranging over the nodes of automata and nodes of the query graph) and the current element type. More specifically, let (q, v, A) be a triplet where q is a certain state of the automaton, v is a node in G_p and $A \in Ele$. We claim that this triplet is the only thing we need to store at each step of the decision algorithm. It is clear that such a triplet only needs logarithmic space. The semantics of a triplet (q, v, A) is as follows: we are in a node that has label A, we are in the query graph G_p in node v and the last visited state of an automaton is q.

We now describe how the NLOGSPACE algorithm works for deciding satisfiability of p in the presence of a DTD D.

Initially, the algorithm stores (q_0, s, r) where is q_0 is a (new) dummy state, s is the input node of G_p and r is the root label.

Assume that the currently stored triplet is (q, v, A). We now define which triplet (q', w', A') will be stored next by the algorithm. Let w be a node in G_p adjacent to v. If multiple such w exist, then we non-deterministically choose one. (Note that the only way a node in G_p can have multiple outgoing edges is in case of union). We now consider the following cases depending on the label of w.

(1) the label of w is ϵ: Then $(q', w', A') = (q, w, A)$.

(2) the label of w is B: Then (q', w', A') is such that $w' = w$, $A' = B$ and q' is a state in M_A that has a B-labeled incoming edge. Again, if multiple such states exist in M_A, we non-deterministically choose one. If no such state exists, then we are in a dead end and we do not have a next triplet.

(3) the label of w is $\downarrow$: Then (q', w', A') is such that $w' = w$, q' is one of the states in M_A not corresponding to the trivial word, and A' is one of the labels of incoming edges in q'. Again, when M_A only produces the empty word then we are in a dead end.

(4) the label of w is $\downarrow^*$: Then (q', w', A') is such that $w' = w$, and we can either take $q' = q$ and $A' = A$ or descend in the tree using the automata. This is done as follows: Select a state q_1 in M_A that does not correspond to the empty word, select an arbitrary incoming edge of q_1 and let A_1 be its label. Next, repeat the same procedure for M_{A_1} and so on. The number of iterations i is chosen non-deterministically (but bounded by the size of the automata). Let q' be the last selected state in M_{A_i} and A' be the label of an incoming edge in q'.

(5) the label of w is $\rightarrow$: Then (q', w', A') is such that $w' = w$ and q' is a state directly connected to q via a labeled edge A'. If no such state exists, we are in a dead end.

(6) the label of w is $\rightarrow^*$: Then (q', w', A') is such that $w' = w$, and q' is a state reachable from q (or q itself) and where A' is the label of last edge leading to q' (or A if we stayed at the state q).

If the algorithm stores (q, t, A), then it decides that p is satisfiable. On the other hand, if p is satisfiable then there exists a run of the algorithm (i.e., a sequence of stored triplets) that ends up in (q, t, A). The last assertion is easy to show. Suppose that $T \models (p, D)$ then the path from the root of T to a node in the answer set $r[\![p]\!]$ can be used as a guide for making the choices in the decision algorithm.

For the other direction, if the algorithm did not end up in a dead end, we can use the successful run of the algorithm to build a tree T such that $T \models (D, p)$. This can easily be shown by induction on the structure of p. The use of automata and states ensures that a sequence of sibling steps result in a word in the corresponding regular language (or a production in the DTD). ∎

Proof of Theorem 2

We show that $\mathsf{SAT}(\mathcal{X}(\rightarrow,\leftarrow))$ is in PTIME by providing a PTIME decision algorithm based on dynamic programming. That is, given (Q, D), where Q is a query in $\mathcal{X}(\rightarrow,\leftarrow)$ and $D = (Ele, Att, P, R, r)$ is a DTD, the algorithm decides whether or not (Q, D) is satisfiable in PTIME.

Observe that any query $Q \in \mathcal{X}(\rightarrow,\leftarrow)$ is of the form $A_1/\eta_1/\ldots/A_n/\eta_n$, where for each $i \in [1, n]$, A_i is an element type, and η_i is a sequence of $\rightarrow$ and $\leftarrow$. Here η_i indicates multi-step horizontal moves and A_i is a one-step downward move. In other words, on an XML tree T, the navigation of Q at the i-th step first moves sideways, and then downward; moreover, as soon as it moves downward, the navigation proceeds to lower levels of T without looking back upward. Note that a query in $\mathcal{X}(\rightarrow,\leftarrow)$ is not satisfiable if it starts with $\rightarrow$ or $\leftarrow$. Let Q_i denote $A_i/\eta_i/\ldots/A_n/\eta_n$, where Q_1 is Q.

We now define the variables to be used in the algorithm. (a) For each Q_i and each $A \in Ele$, we define a Boolean variable $\mathsf{sat}(Q_i, A)$ that indicates whether or not $v[\![Q_i]\!]$ is empty at a context node v of A type, i.e., whether or not Q_i is satisfiable at an A element. (b) For each $A \in Ele$, let M_A be an NFA representing the regular expression $P(A)$, such that each state of M_A is reachable from the start state, and there exists no ϵ-transition. Such an M_A can be computed in PTIME, and the size $|M_A|$ of the NFA is linear in the size $|P(A)|$. For each η_i and each $B \in Ele$, let $\mathsf{reach}(M_A, B, \eta_i)$ be the set consisting of element types C such that there exist two states q_1, q_2 in M_A and (i) there exists an outgoing transition from q_1 labeled B; and (ii) q_2 can be reached from q_1 via B/η_1 ($\rightarrow$ for forward move and $\leftarrow$ for backward); and (iii) the last transition is via an edge labeled C. It is easy to verify that $\mathsf{reach}(M_A, B, \eta_i)$ can be computed in PTIME by, e.g., representing M_A as a graph with inverse edges (for $\leftarrow$), and computing $\mathsf{reach}(M_A, B, \eta_i)$ based on dynamic programming.

Observe that $\mathsf{sat}(Q_i, A)$ is true iff (a) $\mathsf{reach}(M_A, A_i, \eta_i)$ is not empty; (b) there exists $B \in \mathsf{reach}(M_A, A_i, \eta_i)$ such that $\mathsf{sat}(Q_{i+1}, B)$ is true.

Using these variables, the decision algorithm works as follows. It first computes $\mathsf{reach}(M_A, B, \eta_i)$ for each $i \in [1, n]$ and all $A, B \in Ele$. Then, for each Q_i in the decreasing order (i.e., for $i = n, n-1, \ldots, 1$), and for each $A \in Ele$, it computes $\mathsf{sat}(Q_i, A)$ as described above. Both steps can be done in PTIME in $|D|$ and $|Q|$. Hence the algorithm is in PTIME. ∎

Proof of Theorem 3

(1) We show that $\mathsf{SAT}(\mathcal{X}([\,]))$ is NP-hard under non-recursive DTDs by reduction from the 3SAT problem. An instance of 3SAT is a well-formed Boolean formula $\phi = C_1 \wedge \cdots \wedge C_n$ of which we want to decide satisfiability. Assume that the variables in ϕ are $x_1, \ldots, x_m$.

Given ϕ, we define a DTD D_1 and a query Q_1 in $\mathcal{X}([\,])$ such that ϕ is satisfiable iff (Q_1, D_1) is satisfiable. The DTD $D_1 = (Ele, Att, P, R, r)$ is defined as follows:

$Ele = \{C_i \mid i \in [1,n]\} \ \cup \ \{r\}$,
P: $r \to (X_1 + X_1'), \ldots, (X_m + X_m')$,
where X_j denotes $(C_{j_1}, \ldots, C_{j_k})$ for all C_{j_i} in which x_j appears,
and X_j' denotes $(C_{j_{1'}}, \ldots, C_{j_{k'}})$ for all $C_{j_{i'}}$ in which $\bar{x}_j$ appears.
$C_j \to \epsilon$
$Att = \emptyset$, $R(A) = \emptyset$ for all $A \in Ele$.

In a nutshell, the production of r codes a truth assignment of $x_1, \ldots, x_m$ as well as which clauses of ϕ are satisfied by the truth assignment. Note that X_j, X_j' are not element types: they are just shorthands for the concatenations of the clauses that they satisfy.

The query Q_1 is simply $\epsilon[\bigwedge_{j \in [1,n]} C_i]$, i.e., no matter what truth assignment is given, for each $i \in [1,m]$, the clause C_i is satisfied by the truth value of some x_j.

One can then easily verify that there exists an XML tree T such that T satisfies Q and T conforms to D if and only if ϕ is satisfiable. Indeed, there is a one-to-one correspondence between XML trees T conforming to D on the one hand, and truth assignments for the x_i variables on the other hand. Moreover, $T \models Q$ iff all clauses appear as the leaves of the tree T, which holds iff the truth assignment corresponding to T makes all clauses true and hence is a solution of ϕ.

(2) We next show that $\mathsf{SAT}(\mathcal{X}(\to, [\,]))$ is NP-hard under fixed, disjunction-free and non-recursive DTDs by reduction from the 3SAT problem. We define a fixed, disjunction-free and non-recursive DTD D_2 and moreover, given any 3SAT instance ϕ, we find a query Q_2 in $\mathcal{X}(\to, [\,])$ such that ϕ is satisfiable iff (Q_2, D_2) is satisfiable.

Define the DTD $D_2 = (Ele, Att, P, R, r)$ as follows:

$Ele = \{r, S, S_0, X, L, C, T\}$,
P: $r \to S_0, (S, X)^*, S_0$,
$X \to S, L, L, S$,
$L \to S, C^*, S$,
$C \to S, T^*, S$,
$C_i \to \epsilon$, /* similarly for S_0, S, T */
$Att = \emptyset$, $R(A) = \emptyset$ for all $A \in Ele$.

An XML tree conforming to D_2 is depicted in Fig. 3. Intuitively, the root of an XML tree of D_2 consists of a list of X elements. As will been seen shortly, these X elements encode variables $x_i, \ldots, x_m$. Below each X element there are two L elements, where the first L element codes the case when the corresponding

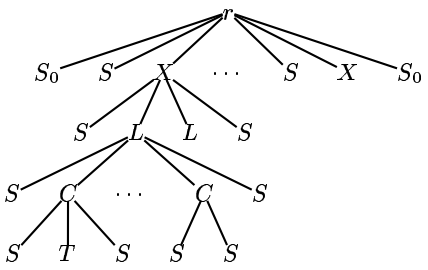

Fig. 3. A tree of the DTD encoding a 3SAT instance in the proof of Theorem 3 (2)

variable is true, and the second L element codes the case when the corresponding variable is false. Below each L is a list of C elements, while each C element may have a T child. The elements S, S_0 serve as delimiters to indicate the start and end of children lists.

Given any instance ϕ of 3SAT as described above, we encode ϕ in terms of a query $Q_2 = \epsilon[q_v \wedge q_c \wedge q_a \wedge q_\phi]$ in $\mathcal{X}(\rightarrow, [\,])$, defined with the following qualifiers at the root.

(a) Encoding variables: $q_v = S_0/{\rightarrow}^{2m}/{\rightarrow}[\mathsf{lab}() = S_0]$, where ${\rightarrow}^{2m}$ is a shorthand for repetition of $\rightarrow$ for $2m$ times. This asserts that the (S, X) list under r consists of precisely m elements of type X. We use X_j to denote $S_0/{\rightarrow}^{2j}$, which encodes the variable x_j in ϕ.

(b) Coding the connection between clauses and literals: $q_c = \bigwedge_{i\in[1,n],j\in[1,m]} (q^T_{i,j} \wedge q^F_{i,j})$, where $q^T_{i,j}, q^F_{i,j}$ are defined as follows:

$$q^T_{i,j} = \begin{cases} X_j/S/{\rightarrow}/S/{\rightarrow}^i/S/{\rightarrow}[\mathsf{lab}() = T] \text{ if } x_j \text{ appears in } C_i \\ X_j/S/{\rightarrow}/S/{\rightarrow}^i/S/{\rightarrow}[\mathsf{lab}() = S] \text{ otherwise} \end{cases}$$

$$q^F_{i,j} = \begin{cases} X_j/S/{\rightarrow}/{\rightarrow}/S/{\rightarrow}^i/S/{\rightarrow}[\mathsf{lab}() = T] \text{ if } \bar{x_j} \text{ appears in } C_i \\ X_j/S/{\rightarrow}/{\rightarrow}/S/{\rightarrow}^i/S/{\rightarrow}[\mathsf{lab}() = S] \text{ otherwise} \end{cases}$$

That is, we encode the clause C_i in terms of C^i under both the first and second L child of X, where C^i is a shorthand for $S/{\rightarrow}^i$, i.e., the i-th C child of L from the left. For each variable x_j (i.e., X_j), if x_j appears (positively) in C_i, then $q^T_{i,j}$ ensures that C^i under the first L has a T child, i.e., C_i is satisfied if x_j is true; if x_j does not appear in C_i, then C^i under the first L has no T child; similarly, $q^F_{i,j}$ encodes the connection between $\bar{x_j}$ (i.e., x_j appears negatively) and C_i. Note that these also assert that below each L there are at least n many C children.

(c) Coding consistent truth assignment: $q_a = \bigwedge_{j\in[1,k]} (X_j[L/S/{\rightarrow}^{n+1}[\mathsf{lab}() = S] \wedge L/S/{\rightarrow}^{n+2}[\mathsf{lab}() = S])$. We encode x_j (i.e., X_j) such that it is assigned true if under the first L child of X_j there are precisely n elements of C type; similarly,

x_j is false if under the second L child of X_j there are n elements of C type. The qualifier q_a asserts that for each x_j there is a single truth value, i.e., only one of the C lists under X^j has n elements of type C.

(d) Encoding clauses: $q_\phi = \bigwedge_{i\in[1,n]} X/L/C^i[T]$. This asserts that all clauses C_i (e.g., C^i) must be satisfied by a truth assignment of some x_j, no matter what x_j is.

One can easily verify that Q is in $\mathsf{SAT}(\mathcal{X}(\rightarrow,[\,]))$ and furthermore, that Q is satisfiable by an XML tree of the fixed DTD D_2 iff the 3SAT instance ϕ is satisfiable.

A mild variation of the proof above suffices to show that $\mathsf{SAT}(\mathcal{X}(\leftarrow,[\,]))$ is NP-hard. Indeed, let Q_2' be defined by substituting '$\leftarrow$' for '$\rightarrow$' in Q_2 and changing X_j to $S_0/{\leftarrow}^{2j-1}$, then one can verify that there exists an XML tree T such that T satisfies Q_2' and T conforms to D_2 if and only if ϕ is satisfiable.

(3) Finally, we show that $\mathsf{SAT}(\mathcal{X}(\rightarrow,\cup,[\,]))$ is NP-hard in the absence of DTDs, by reduction from the 3SAT problem.

In a nutshell, given an instance ϕ of 3SAT as described above, we define a query Q_3 in $\mathcal{X}(\rightarrow,\cup,[\,])$ to encode ϕ, and moreover, we will constrain XML trees that satisfy Q_3 by forcing them to contain a consecutive sequence of children whose labels form the string $XZ_1XZ_2\cdots XZ_m$, where m is the number of variables in ϕ and $Z_i \in \{T,F\}$ for $i \in [1,m]$. The idea is that the ith Z_i child (which equals T or F) in such a sequence denotes the truth value of the i-th variable x_i in ϕ.

We will use the following qualifiers to encode ϕ:

(a) Encoding the clauses. For each clause $C_j = l_1 \vee l_2 \vee l_3$, we let $q_j = xp(l_1) \vee xp(l_2) \vee xp(l_3)$, where $xp(l_i) = {\rightarrow}^{2j-1}[\mathsf{lab}() = T]$ if $l_i = x_j$ and $xp(l_i) = {\rightarrow}^{2j-1}[\mathsf{lab}() = F]$ if $l_i = \bar{x}_j$.

(b) Let $p_m = X(\bigwedge_{i=1}^{m-1} {\rightarrow}^{i+1}[\mathsf{lab}() = X])$. This expression selects X children that have $(m-1)$ right X siblings, where consecutive X siblings are separated by a single sibling.

Taken together, the final $\mathcal{X}(\rightarrow,\cup,[\,])$ query Q_3 encoding ϕ is now

$$Q_3 = \epsilon\big[X[(\bigwedge_{i=1}^{m-1} {\rightarrow}^{i+1}[\mathsf{lab}() = X]) \wedge (q_1 \wedge \cdots \wedge q_n)]\big].$$

One can easily verify that Q_3 is satisfiable iff ϕ is satisfiable.

Similarly, one can verify that $\mathsf{SAT}(\mathcal{X}(\leftarrow,\cup,[\,]))$ is also NP-hard in the absence of DTDs ∎

Proof of Theorem 4

We show that $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,\uparrow,\uparrow^*,\leftarrow,\leftarrow^*,\rightarrow,\rightarrow^*,\cup,[\,],=))$ is in NP in the presence of DTDs. From Proposition 5, we know that it is sufficient to show that $\mathsf{SAT}(\mathcal{X}^*(\downarrow,\downarrow^*,\uparrow,\uparrow^*,\cup,[\,],=))$ is in NP in the presence of DTDs describing a binary tree. Observe that we may assume that p does not contain $\cup$. Indeed, for each $\cup$ in p we can guess non-deterministically one of the alternatives and in this way get rid of the unions. So, let $p \in \mathcal{X}^*(\downarrow,\downarrow^*,\uparrow,\uparrow^*,[\,],=)$ and let D be a DTD.

The proof is similar to the proof that $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,\uparrow,\uparrow^*,[\,],=))$ is in NP [2, Theorem 4.5]. It consists of the following steps: we translate $T \models p$ in terms of a so-called skeleton T_p associated with p into T. We call such an embedded skeleton a witness skeleton. More specifically, we show that $T \models p$ if and only if T contains a witness skeleton $\gamma(T_p)$, where γ denotes the embedding. The main observation is then that if there exists a tree T conforming to D and a witness skeleton for p in T, then there exists a tree T' conforming to D and a witness skeleton for p in T' whose depth is polynomial in the size of p and D. The proof of this observation is based on a short-cutting technique. Once we established a bound on the depth of the witness skeleton for p, we show that we can guess a candidate witness skeleton for p using D and verify whether it is a real witness skeleton in PTIME.

We only describe the differences from the proof of Theorem 4.5 presented in [2] and refer to the full version of the paper for the detailed proof.

Skeletons and Witnesses. For a given p we define the skeleton T_p inductively on the structure of p as described in [2]. We only need to describe how we deal with the Kleene star. Denote by $p[i_1,\ldots,i_m]$ the XPath expression obtained by replacing the Kleene stars with exponents $i_1,\ldots,i_m$ (assuming we have m Kleene stars). As a result, $p[i_1,\ldots,i_m] \in \mathcal{X}(\downarrow,\downarrow^*,\uparrow,\uparrow^*,[\,],=)$.

Using the definition of skeleton described in [2], we end up with a skeleton $T_{p[i_1,\ldots,i_m]}$. Having defined the skeleton of an XPath expression, we explain now what it means to have a witness skeleton for p. Let T be an arbitrary tree and let $T_{p[i_1,\ldots,i_m]}$ denote the skeletons for $p[i_1,\ldots,i_m]$. We say that T contains a witness skeleton of p if there exist m integers $i_1,\ldots,i_m$ and a mapping $\gamma : T_{p[i_1,\ldots,i_m]} \rightarrow T$ satisfying the conditions for being an embedding as stated in [2]. If such a sequence of integers and an embedding γ exist, we denote this by $T_{p[i_1,\ldots,i_m]} \subseteq_\gamma T$. It follows directly from the semantics of XPath queries that

$$T \models p \text{ if and only if there exists } i_1,\ldots,i_m \text{ and } \gamma \text{ such that } T_{p[i_1,\ldots,i_m]} \subseteq_\gamma T$$

Short-cutting For integers $i_1,\ldots,i_m$, let the depth of a witness skeleton for p in T be the depth in T of the deepest node in $\gamma(T_{p[i_1,\ldots,i_m]})$. In [2], it has been shown that if there exists a witness skeleton for $p[i_1,\ldots,i_m]$ in T, then there

exists a tree T' and witness skeleton for $p[i_1, \ldots, i_m]$ in T' of depth bounded by a polynomial in $|p[i_1, \ldots, i_m]|$ and $|D|$.

We now observe that it is sufficient to choose the integers $i_1, \ldots, i_m$ from the interval $[0, |Ele|]$. Consider the case of a single Kleene star. Suppose that we choose an integer k larger than $|Ele|$, then due to the special form of our Kleene star expression, we can designate $k+1$ nodes $\gamma(n_i)$ that correspond to the root nodes of the copies of the skeleton of the expression within the Kleene star. Since $k > |Ele|$ at least two node $\gamma(n_i)$ and $\gamma(n_j)$, with $i \leq |Ele| < j$, must have the same element type, and hence we could have taken $k = i < |Ele|$. For multiple Kleene stars, we observe that we do not have nested Kleene stars and hence we can treat each Kleene star independently.

Guessing and verifying. Clearly, a witness skeleton for $p[i_1, \ldots, i_m]$ in T induces a subtree of T by looking at all the paths from the root of T to the deepest witness nodes. We call this tree the witness tree of $p[i_1, \ldots, i_m]$ in T. We now first guess m integers $i_1, \ldots, i_m \in [0, |Ele|]$ and then apply the guessing and PTIME verification procedure as described in [2]. ∎

Proof of Theorem 5

(1) We first show that $\mathsf{SAT}(\mathcal{X}(\rightarrow, [\,], \neg))$ is PSPACE-hard under no-star and non-recursive DTDs by reduction from 3QSAT. An instance of the Q3SAT problem consists of a well-formed quantified Boolean formula $\phi = Q_1 x_1 Q_2 x_2 \cdots Q_n x_n E$, where $E = C_1 \wedge \cdots \wedge C_m$ is an instance of 3SAT in which all the propositional variables are $x_1, \ldots, x_n$, and $Q_i \in \{\forall, \exists\}$ for each $i \in [1, n]$. The Q3SAT problem is to decide, given such a quantified Boolean formula ϕ, whether or not ϕ is true.

Given a quantified Boolean formula ϕ as described above, we encode ϕ in terms of a query Q_ϕ in $\mathcal{X}(\rightarrow, [\,], \neg)$ and a no-star, non-recursive DTD D such that ϕ is satisfiable iff (Q_ϕ, D) is satisfiable.

We define a no-star and non-recursive DTD $D = (Ele, Att, P, R, r)$ as follows:

$$
\begin{array}{ll}
Ele = \{X, T, F, S\} & \\
P: \quad r \rightarrow X, \ldots, X & /^*\ n \text{ occurrences of } X\ ^*/ \\
\quad X \rightarrow S, (T + \epsilon), (F + \epsilon), & \\
\quad T \rightarrow \epsilon, & \\
\quad F \rightarrow \epsilon, & \\
\quad S \rightarrow \epsilon, & \\
Att = \emptyset, \quad R(A) = \emptyset \text{ for all } A \in Ele. &
\end{array}
$$

In a nutshell, the i-th X child of the root is to code the variable x_i, which has a truth assignment T or F.

We next define Q_ϕ to be $\epsilon[q_1 \wedge q_2]$, where q_1 is to code the quantifiers $Q_1x_1Q_2x_2\cdots Q_nx_n$ in ϕ, and q_2 is to code the the 3SAT instance $E = C_1 \wedge \cdots \wedge C_m$. More specifically, $q_1 = \bigwedge_{i\in[1,n]} q^i$, where $q^i = \rightarrow^i[S/\rightarrow/\rightarrow]$ if Q_i is '$\forall$', and $q^i = \rightarrow^i[S/\rightarrow[\neg\rightarrow]]$ if Q_i is '$\exists$'; here $\rightarrow^i$ denotes the repetition of '$\rightarrow$' for i times. That is, if x_i is universally quantified, then both T and F values of x_i should be considered; otherwise only one of these truth values is needed.

We define q_2 to be $(\neg c_1 \wedge \ldots \wedge \neg c_m)$, where c_i represents *the negation* of the clause C_i. More specifically, let $C_i = l_i^1 \vee l_i^2 \vee l_i^3$, where l_i^j is a literal, i.e., it is either a variable x_i or the negation $\bar{x}_i$ of a variable. Then c_i is to code $\neg l_i^1 \wedge \neg l_i^2 \wedge \neg l_i^3$. Without loss of generality, assume that the variables of these literals are x_k, x_l and x_m, respectively, with $k < l < m$. Then c_i is defined as

$$c_i = \rightarrow^k/[X/Z_k]/\rightarrow^{l-k}[X/Z_l]\rightarrow^{m-l}/[X/Z_m],$$

where $Z_j = F_j$ if x_j appears in c_i, and $Z_j = T_j$ if $\bar{x}_j$ appears in c_i, for j ranging over k, l, m. For example, if $C_i = x_k \vee \bar{x}_l \vee x_m$ with $k < l < m$, then $c_i = \rightarrow^k[X/F]/\rightarrow^{l-k}[X/T]/\rightarrow^{m-l}[X/F]$. Intuitively, $\neg c_i$ is to assert that for all possible truth assignments consistent with q_1, the clause C_i is true.

It is easy to verify that (Q_ϕ, D) is satisfiable by an XML tree T if and only if the quantified Boolean formula ϕ is true. Similarly, it can be shown that $\mathsf{SAT}(\mathcal{X}(\leftarrow, [\,], \neg))$ is PSPACE-hard under no-star and non-recursive DTDs.

(2) We next show that $\mathsf{SAT}(\mathcal{X}(\rightarrow, [\,], \neg))$ is PSPACE-hard in the absence of DTDs. A mild variation of the proof of Proposition 3 suffices to show that the DTD D above can be coded in terms of a qualifier Q_D in $\mathsf{SAT}(\mathcal{X}(\rightarrow, [\,], \neg))$. Note that Q_D does not use $\downarrow^*$ since D is non-recursive. Thus the PSPACE-hardness result holds for $\mathcal{X}(\rightarrow, [\,], \neg)$ in the absence of DTDs.

Similarly we can show that $\mathsf{SAT}(\mathcal{X}(\leftarrow, [\,], \neg))$ is PSPACE-hard in these settings. ∎

Proof of Theorem 6

We show that $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \leftarrow, \leftarrow^*, \rightarrow, \rightarrow^*, \cup, [\,], \neg))$ is in PSPACE under no-star DTDs using the PSPACE-completeness result for $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \cup, [\,], \neg))$ established in [2]. The lower bound has been shown by Theorem 5 above.

Let $D = (Ele, Att, P, R, r)$ be a no-star DTD. Since D has no Kleene star, we have an upper bound on the number of children in each node in any tree T conforming to D. We denote this upper bound by $cmax(D)$.

Let $p \in \mathcal{X}(\downarrow, \uparrow, \leftarrow, \leftarrow^*, \rightarrow, \rightarrow^*, \cup, [\,], \neg))$. From Proposition 5 and its proof, we know that p can be rewritten into $f(p) \in \mathcal{X}^*(\downarrow, \uparrow, \cup, [\,], \neg)$. Since the horizontal

relation between siblings is encoded vertically by means of labels, we need (a limited form of) conditional path expressions to encode the navigational axis. However, since we have a bound on the number of children and our XPath fragment does not allow for descendant or ancestor, we can replace the Kleene star expressions of the form $(\downarrow[\mathsf{lab}() = A])^*$ in $f(p)$ (given in the proof of Proposition 5) by

$$\cup_{k=1}^{cmax(D)}(\downarrow[\mathsf{lab}() = A])^k.$$

Consequently, $f(p)$ can be rewritten into an expression $p \in \mathcal{X}(\downarrow, \uparrow, \cup, [\ ], \neg)$, a fragment for which the PSPACE upper bound already holds [2]. ∎

Proof of Theorem 7

We show that $\mathsf{SAT}(\mathcal{X}(\uparrow, \rightarrow, [\ ], \neg))$ is EXPTIME-hard under fixed and non-recursive DTDs by reduction from the two-player game of corridor tiling (TPG-CT). To simplify the discussion we give the reduction in terms of a query in $\mathcal{X}(\uparrow, \rightarrow, \cup, [\ ], \neg)$. However, union and disjunction can always eliminated by replacing $p_1 \cup p_2$ by $\epsilon[\neg(\neg p_1 \wedge \neg p_2)]$.

PROBLEM: TPG-CT (Two-Player Game of Corridor Tiling)

INPUT: A tiling system $(X, H, V, \boldsymbol{t}, \boldsymbol{b})$ and a natural number n, where X is a finite set of dominoes (tiles), $H, V \subseteq X \times X$ are two binary relations, $\boldsymbol{t}$ and $\boldsymbol{b}$ are two n-vectors of given tiles in X, and n is the number of columns (the width of the corridor).

QUESTION: Does player I have a winning strategy for tiling the corridor? By tiling the corridor we mean that there exists a tiling $\tau : \mathbb{N} \times \mathbb{N} \rightarrow X$ and a natural number m such that for all $x \in [1, n]$ and $y \in [1, m]$, the *tiling adjacency conditions* are observed, i.e.,

- if $\tau(x, y) = d$ and $\tau(x + 1, y) = d'$ then $(d, d') \in H$;
- if $\tau(x, y) = d$ and $\tau(x, y + 1) = d'$ then $(d, d') \in V$;
- $\tau(x, 1) = \boldsymbol{t}[x]$ and $\tau(x, m) = \boldsymbol{b}[x]$, where $\boldsymbol{t}[x]$ (resp. $\boldsymbol{b}[x]$) denotes the x-th element of the vector $\boldsymbol{t}$ (resp. $\boldsymbol{b}$); that is, the given tiles of $\boldsymbol{t}$ and $\boldsymbol{b}$ are placed in the top and bottom rows, respectively.

Player I and Player II in turn place a tile from X in the first free location (row by row from left to right), observing the tiling adjacency conditions. The one who is unable to make the next move loses the game. Player II may also decide to end the game and check the tiling adjacency conditions at the bottom row. The given tiles $\boldsymbol{t}$ and $\boldsymbol{b}$ are placed in the top and bottom rows by the referee of the game.

Given an instance, $(X, H, V, \boldsymbol{t}, \boldsymbol{b})$ and n, of TPG-CT, we define a fixed and non-recursive DTD D and an XPath query Q in $\mathcal{X}(\uparrow, \rightarrow, [\ ], \neg)$ such that there exists

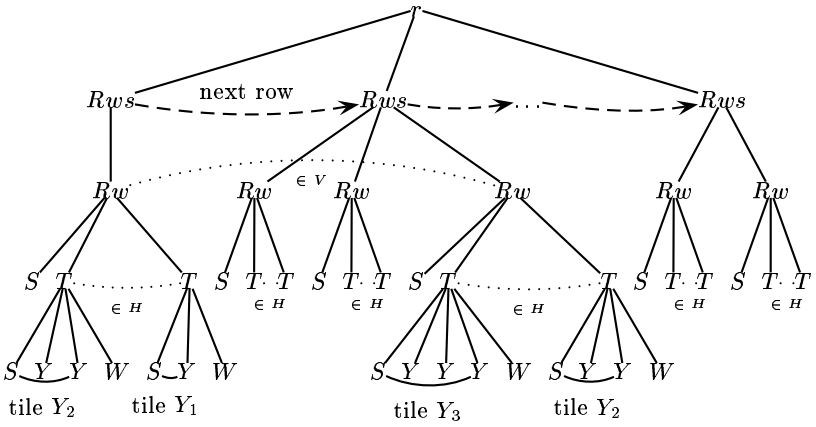

Fig. 4. Illustration of the DTD used in the EXPTIME-hardness proof of $\mathsf{SAT}(\mathcal{X}(\uparrow, \rightarrow, [\,], \neg))$.

an XML tree T satisfying (Q, D) if and only if Player I has a winning strategy for the game. Let $X = \{x_1, \ldots, x_k\}$. Without loss of generality, we assume that the n is even and Player I moves first; the coding for an odd n is similar.

We define the fixed and non-recursive DTD $D = (Ele, Att, P, R, r)$ as follows:

$$
\begin{array}{ll}
Ele = \{r,\ Rws,\ Rw,\ T,\ Y,\ S\}. & \\
P: & r \rightarrow Rws^* \\
 & Rws \rightarrow Rw^* \\
 & Rw \rightarrow S, T^* \\
 & T \rightarrow S, Y^*, W^* \\
Att = \emptyset, & R(A) = \emptyset \text{ for all } A \in Ele.
\end{array}
$$

An XML tree of D consists of a root node with a node Rws for each row in the tiling. Each Rws node will have a bunch of children Rw representing all possible tilings of that row. Each Rw node in its turn has a T node for each tile. Each tile Y_i is coded in terms of exactly i many Y children. Here W is a label indicating whether the tiling is part of row, where each tile has a W label, and such that there exists a row below and above satisfying the vertical constraints. We have depicted (part of) an instance of such an XML tree in Figure 4.

We now use XPath qualifiers in $\mathcal{X}(\uparrow, \rightarrow, \cup, [\,], \neg)$ to encode the following.

(1) Tile type. We often need to express that the tile below T corresponds to tile Y_i. We express this as tile_{Y_i}, which is the shorthand for

$$\mathsf{tile}_{Y_i} = S[\rightarrow^i[\mathsf{lab}() = Y] \wedge (\rightarrow^{(i+1)}[\mathsf{lab}() \neq Y] \vee \neg(\rightarrow^{(i+1)}))].$$

(2) Initial values. There is a Rws node consisting of a single Rw node below which the initial tile configuration $\boldsymbol{t}$ is encoded. This is stated as follows:

$$Q_0 = Rws[(\neg Rw[\rightarrow]) \wedge \bigwedge_{i\in[1,n]} Rw/S[\rightarrow^i[\mathsf{tile}_{\boldsymbol{t}_i} \wedge W]]].$$

(3) Cardinality conditions. Below each Rw node we need to ensure that there are exactly n tiles, i.e., n children of type T. We express this by the qualifier

$$Q_{\text{card}} = \neg Rws/Rw[\bigvee_{i\in[1,n-1]} S[\rightarrow^{(i)} \wedge \neg(\rightarrow^{(i+1)})] \vee S[\rightarrow^{(n+1)}]].$$

(4) Horizontal constrains: Below each Rw node, consecutive T children must satisfy the horizontal tiling constraints. We state this as follows:

$$Q_H = \neg Rws/Rw[\bigvee_{i\in[1,n-1]} \bigwedge_{(x,x')\in H} (S/\rightarrow^i[\mathsf{tile}_x] \wedge \neg S/\rightarrow^{(i+1)}[\mathsf{tile}_{x'}])].$$

(5) Vertical constraints: For any two consecutive Rws nodes r_1 and r_2, every Rw node s_1 below r_1 that has a W child must have a corresponding Rw nodes s_2 below r_2 that has a W child such that the tilings below s_1 and s_2 satisfy the vertical tiling constraints. In other words, there does not exist a Rws with a Rw node that has no "matching" (with respect to the vertical constraints) Rw node in the consecutive Rws node. We can express this as follows:

$$Q_V = \neg Rws/Rw[\bigvee_{(x,y)\in V} \bigvee_{i\in[1,n]} (S[\rightarrow^i[\mathsf{tile}_x \wedge W] \wedge \neg\uparrow/\rightarrow/Rw[\neg S[\rightarrow^i[\mathsf{tile}_y \wedge W]]])].$$

(6) Play continues unless Player I has won. Unless there is a node Rws below which the bottom vector $\boldsymbol{b}$ appears in one of its Rw nodes, there is always a right sibling to Rws, i.e., the game continues. We express this as a qualifier Q_p:

$$Q_p = \neg Rws[Rw[\bigwedge_{i\in[1,n]} S/\rightarrow^i[\mathsf{tile}_{\boldsymbol{b}[i]} \wedge W] \wedge (\uparrow/\rightarrow)]].$$

(7) Player I responds to all possible moves of Player II. Suppose that Player I just placed tile x at position i (odd), and that this row is represented by a Rw node v, then for any tile x' that satisfies the horizontal constraint $(x, x') \in H$, the immediate right Rw sibling of the Rw node v represents this possible tiling. More specifically, for odd i,

$$Q_i = \neg Rws[Rw[\bigvee_t (S/\rightarrow^i[\mathsf{tile}_t] \wedge \uparrow/\rightarrow/[\neg \bigvee_{(t,s)\in H} (S/\rightarrow^i[\mathsf{tile}_t] \wedge S/\rightarrow^{i+1}[\mathsf{tile}_s])])]]$$

We then consider $Q_a = \bigwedge_{i=1}^{n/2} Q_{2i-1}$, This qualifier forces that below any Rws, all possible tiles of a row appear. Note that some of them will not be consistent with the vertical adjacency conditions. However, each Rw that represents a tiling of a row consistent with vertical conditions as well will have a W descendant (as is forced by the Q_V qualifier).

(8) Correct W labeling. It is not sufficient to have some tiles in a row that have a W child. We require that either each tile in a row has a W child, or none of the tiles has. This can be easily expressed as follows.

$$Q_W \;=\; \neg Rws/Rw[\bigvee_{i=1}^{n} S/{\rightarrow}^i[W] \wedge \bigvee_{i\in[1,n-1]} S[{\rightarrow}^{(i)}[W] \wedge \neg({\rightarrow}^{(i+1)})[W]]].$$

Finally, we obtain the qualifier

$$Q = Q_0 \wedge Q_{\text{card}} \wedge Q_H \wedge Q_V \wedge Q_p \wedge Q_a \wedge Q_W.$$

It is easily verified that Player I has a winning strategy if and only if there is an XML document T that represents the game tree such that $T \models (Q, D)$. ∎

Proof of Theorem 8

(1) We prove the undecidability of $\mathsf{SAT}(\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg))$ in the presence of fixed, non-recursive DTDs by reduction from the halting problem for two-register machines, which is known to be undecidable (see, e.g., [4]).

Two Register Machine. A two-register machine (2RM) M has two registers $\mathsf{register}_1, \mathsf{register}_2$, and is programmed by a numbered sequence $I_0, I_1, ..., I_l$ of instructions. Each register contains a natural number. An *instantaneous description* (ID) of M is (i, m, n), where $i \in [0, l]$, m and n are natural numbers. It indicates that M is to execute instruction I_i (or at "state i") with $\mathsf{register}_1$ and $\mathsf{register}_2$ containing m and n, respectively.

An instruction I_i of M can be either an *addition* or a *subtraction*, which defines a relation $\rightarrow_M$ between IDs, described as follows:

- *addition*: (i, rg, j), where rg is either $\mathsf{register}_1$ or $\mathsf{register}_2$, and $0 \leq i, j \leq l$. Its semantics is: at state i, M adds 1 to the content of rg, and then goes to state j. Accordingly:

$$(i, m, n) \rightarrow_M \begin{cases} (j, m+1, n) \text{ if } \mathsf{rg} = \mathsf{register}_1 \\ (j, m, n+1) \text{ otherwise} \end{cases}$$

- *subtraction*: (i, rg, j, k), where rg is either $\mathsf{register}_1$ or $\mathsf{register}_2$, and $0 \leq i, j, k \leq l$. Its semantics is: at state i, M tests whether the content of rg

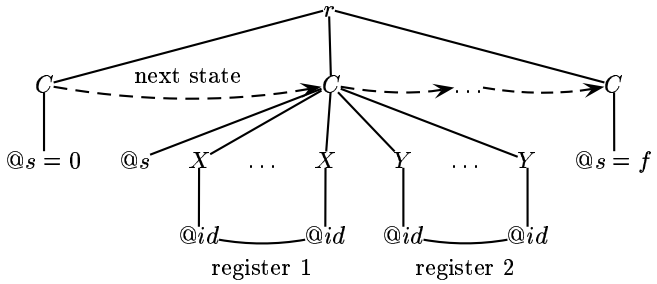

Fig. 5. Illustration of the DTD used in the undecidability proof of $\mathsf{SAT}(\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg))$.

is 0, and if it is, then goes to state j; otherwise M subtracts 1 from the content of rg and goes to the state k. Accordingly:

$$(i,m,n) \rightarrow_M \begin{cases} (j,0,n) & \text{if } \mathsf{rg} = \mathsf{register}_1 \text{ and } m = 0 \\ (k,m-1,n) & \text{if } \mathsf{rg} = \mathsf{register}_1 \text{ and } m \neq 0 \\ (j,m,0) & \text{if } \mathsf{rg} = \mathsf{register}_2 \text{ and } n = 0 \\ (k,m,n-1) & \text{if } \mathsf{rg} = \mathsf{register}_2 \text{ and } n \neq 0 \end{cases}$$

Assume, w.l.o.g., that the initial ID is $I = (0,0,0)$ and that the final ID is $F = (f,0,0)$, i.e., a halting state $f \in [0,l]$ with zeros in both registers. The *halting problem for 2RM* is to determine, given a 2RM M as described above, whether or not $I \Rightarrow_M F$, where $\Rightarrow_M$ denotes the reflexive and transitive closure of $\rightarrow_M$.

Reduction. We now provide reduction from the halting problem for 2RM to $\mathsf{SAT}(\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg))$. Given an instance M of 2RM as described above, we encode M in terms of a DTD D and an XPath query $p \in \mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg)$.

More specifically, we define the DTD $D = (Ele, Att, P, R, r)$ as follows:

$Ele = \{r,\ C,\ X,\ Y\}$.
P: $\quad r \rightarrow C^*$
$\qquad C \rightarrow X^*, Y^*$.
$Att = \{@s,\ @id\}, \quad R(C) = \{@s\},\ R(X) = R(Y) = \{@id\}$.

As depicted in Figure 5, an XML tree of the DTD D consists of an unbounded number of C children coding the IDs of M executing starting from the outer

leftmost C element. Each C element encodes and ID of the 2RM M: it has an s attribute indicating the state of the ID, and it has a number of X children on the left followed by a number of Y children on the right. The number of X (resp. Y) children represents the contents of $\mathsf{register}_1$ (resp. $\mathsf{register}_2$). To count the number of X children with the same parent, an $X.id$ attribute is defined for X elements, which is to serve as a *local key* for the X elements in a block. Similarly for the Y below the same parent.

We also use the following qualifiers at the root to encode the 2RM M, expressed in $\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg)$.

(1) *Initial ID.* We code the initial ID $(0, 0, 0)$ of M by using the leftmost C child of the root r.

$$Q_{start} \quad = \quad C[(\epsilon/@s = 0 \ \wedge \ [\neg(\leftarrow \vee X \vee Y)])].$$

(2) *Halting state.* The final ID $(f, 0, 0)$ of M is expressed as

$$Q_{halting} \quad = \quad C[(\epsilon/@s = f \ \wedge \ [\neg(\rightarrow \vee X \vee Y)])].$$

This asserts that there exists an ID (i.e., a C child of r) such that it is $(f, 0, 0)$, and moreover, no more computation is conducted (or ID is reached) after M enters $(f, 0, 0)$.

(3) *Local key.* To count the number of X children (resp. Y children) under a C element, we enforce that $X.id$ is a local key for the set of X children (resp. Y children):

$$Q_{xKey} = \neg C[X/@id = \rightarrow/\rightarrow^*[\mathsf{lab}() = X]/@id],$$
$$Q_{yKey} = \neg C[Y/@id = \rightarrow/\rightarrow^*[\mathsf{lab}() = Y]/@id].$$

This suffices as it asserts that the $X.id$ of any X is different from the $X.id$ of any of its X siblings; in other words, all the X children of C have distinct $X.id$ values; similarly for Y elements.

(4) *Transition.* For each $i \in [0, l]$, we code the i-th instruction I_i with a qualifier Q_i, based on the type of I_i.

(Case 1: *Addition*). If I_i is an *addition* transition (i, rg, j), where $\mathsf{rg} = \mathsf{register}_1$, then Q_i is defined to ensure that *for any* C element c_1 with state $c_1.s = i$, (i) its right sibling c_2 has state $c_2.s = j$ (state change); (ii) c_2 has one more X child than c_1 ($\mathsf{register}_1$ is incremented by 1); and moreover, (iii) c_2 has the same number of Y children as c_1 ($\mathsf{register}_2$ remains unchanged). These are expressed as follows:

$$
\begin{aligned}
Q_i &= \neg C[\epsilon/@s = i \ \wedge (\rightarrow/@s \neq j \ \vee \ Q_a^X \ \vee \ Q^Y)] \\
Q_a^X &= (X[\neg(\epsilon/@id = \uparrow/\rightarrow/X/\rightarrow^*[\mathsf{lab}() = X]/@id)]) \\
&\quad \vee (\rightarrow/X/\rightarrow^*[\mathsf{lab}() = X \wedge \rightarrow[\mathsf{lab}() = X] \\
&\quad\quad \wedge \neg(\epsilon/@id = \uparrow/\leftarrow/X/\rightarrow^*[\mathsf{lab}() = X]/@id)]) \\
Q^Y &= (Y[\neg(\epsilon/@id = \uparrow/\rightarrow/Y/\rightarrow^*/@id)]) \\
&\quad \vee (\rightarrow/Y[\neg(\epsilon/@id = \uparrow/\leftarrow/Y/\rightarrow^*/@id)]).
\end{aligned}
$$

Here Q_i, Q_a^X and Q^Y assert the conditions (i – iii) above. Similarly, Q_i, Q_a^X and Q^Y can be defined for $\mathsf{rg} = \mathsf{register}_2$.

(Case 2: *Subtraction*). If I_i is a subtraction (i, rg, j, k), where $\mathsf{rg} = \mathsf{register}_1$, then Q_i is defined to ensure that *for any* C element c_1 with state $c_1.s = i$, (i) its right sibling c_2 has state $c_2.s = j$ if c_1 has no X child (i.e., $\mathsf{register}_1$ is 0), and furthermore, c_2 has no X child and it has the same number of Y children as c_1; in other words, the contents of registers remain unchanged; and (ii) if c_1 has an X child (i.e., $\mathsf{register}_1 \neq 0$), then c_2 has state $c_2.s = k$, and moreover, c_2 has one less X child than c_1 ($\mathsf{register}_1$ is decremented by 1), while the number of Y children of c_2 is the same as that of c_1 ($\mathsf{register}_2$ remains unchanged). These are expressed as follows:

$$
\begin{aligned}
Q_i &= \neg C[\epsilon/@s = i \ \wedge (Q_s^0 \ \vee \ Q_s^X)] \\
Q_s^0 &= (\neg X \ \wedge \ (\rightarrow/@s \neq k \ \vee \ \rightarrow[X] \ \vee \ Q^Y)) \\
Q_s^X &= (X \ \wedge \ (\rightarrow/@s \neq j \\
&\quad \vee X[\rightarrow^*[\mathsf{lab}() = X \wedge \rightarrow[\mathsf{lab}() = X] \\
&\quad\quad \wedge \neg(\epsilon/@id = \uparrow/\rightarrow/X\rightarrow^*[\mathsf{lab}() = X]/@id)]) \\
&\quad\quad \vee (\rightarrow/X[\neg(\epsilon/@id = \uparrow/\leftarrow/X/\rightarrow^*[\mathsf{lab}() = X]/@id)])
\end{aligned}
$$

Here Q^Y is the same as defined in Case 1, and Q_i, Q_s^0 and Q_s^X assert the conditions (i – ii) above. Similarly, Q_i, Q_s^0 and Q_s^X can be defined for $\mathsf{rg} = \mathsf{register}_2$.

Putting these together, we define the query p to be

$$
\epsilon[Q_{start} \ \wedge \ Q_{halting} \ \wedge \ Q_{xKey} \ \wedge \ Q_{yKey} \ \wedge \ \bigwedge_{i \in [0,l]} Q_i].
$$

One can verify that p is in $\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\ ], =, \neg)$ and furthermore, that (D, p) is satisfiable iff the 2RM M halts, i.e., $(0,0,0) \Rightarrow_M (f,0,0)$.

(2) We next show that $\mathsf{SAT}(\mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\ ], =, \neg))$ is undecidable in the absence of DTDs. Consider the DTD D given in part (1). We will encode this DTD using qualifiers.

First, the production $P(r)$ can be easily expressed by a qualifier saying that there should be a C child of the root, followed only by C labeled right siblings:

$$
Q_r = C[\neg(\leftarrow) \wedge \neg(\rightarrow^*[\mathsf{lab}() \neq C])].
$$

For the production $P(C)$ we express that only X and Y labeled children appear and no X labeled node can come after an Y labeled node. It is easier to encode the production $P(C) = S, X^*, Y^*$ since then we know that C has at least one child which we can use to go down (we do not have $\downarrow$ in the XPath fragment). It can be easily verified that the undecidability proof of (1) goes through after these minor modifications.

We then encode (the modified) $P(C)$ with a qualifier saying that there should be a single leftmost S child below C, followed by only X and Y labeled right siblings in the right order (Y after X):

$$Q_C = \neg C[S[\leftarrow \vee \rightarrow/\rightarrow^*[\mathsf{lab}() \neq X \wedge \mathsf{lab}() \neq Y]] \wedge \neg S] \wedge \\ \neg C[S/\rightarrow/\rightarrow^*[\mathsf{lab}() = Y]/\rightarrow/\rightarrow^*[\mathsf{lab}() = X]].$$

For a given $p \in \mathcal{X}(\uparrow, \leftarrow, \rightarrow, \rightarrow^*, \cup, [\,], =, \neg)$, it is then easy to see that $T \models (p, D)$ iff $T \models \epsilon[Q_r \wedge Q_C]/p$. ∎

Proof of Proposition 8

The proof of the first statement is the same as the proof of Proposition 3.3 in [2]. For the second statement, consider fragments X that contains negation and $\uparrow$ and is closed under inverse. Given any DTD D and any queries $p_1, p_2 \in \mathcal{X}$, it is easy to verify, by induction on the structure of p_2, (a) $p_1 \subseteq p_2$ under D iff $p = p_1[\neg(\mathsf{inverse}(p_2)[\neg\uparrow])]$ is satisfiable, where $\mathsf{inverse}$ is an extension of that function given in [2], as described in Section 5, and the qualifier $[\neg\uparrow]$ conducts the root test; and (b) $\mathsf{inverse}(p_2)$ is computable in $O(|p_2|)$ time. Thus $\mathsf{CNT}(\mathcal{X})$ can be reduced to $\mathsf{SAT}(\mathcal{X})$ in linear time. ∎