XPath Satisfiability in the Presence of DTDs

Michael Benedikt
Bell Laboratories
benedikt@research.bell-labs.com

Wenfei Fan
Univ. of Edinburgh & Bell Labs
wenfei@inf.ed.ac.uk

Floris Geerts
Univ. of Limburg & Univ. of Edinburgh
fgeerts@inf.ed.ac.uk

ABSTRACT

We study the satisfiability problem associated with XPath in the presence of DTDs. This is the problem of determining, given a query p in an XPath fragment and a DTD D, whether or not there exists an XML document T such that T conforms to D and the answer of p on T is nonempty. We consider a variety of XPath fragments widely used in practice, and investigate the impact of different XPath operators on satisfiability analysis. We first study the problem for negation-free XPath fragments with and without upward axes, recursion and data-value joins, identifying which factors lead to tractability and which to NP-completeness. We then turn to fragments with negation but without data values, establishing lower and upper bounds in the absence and in the presence of upward modalities and recursion. We show that with negation the complexity ranges from PSPACE to EXPTIME. Moreover, when both data values and negation are in place, we find that the complexity ranges from NEXPTIME to undecidable. Finally, we give a finer analysis of the problem for particular classes of DTDs, exploring the impact of various DTD constructs, identifying tractable cases, as well as providing the complexity in the query size alone.

1. INTRODUCTION

XPath [6] has been widely used in XML query languages (e.g., XSLT, XQuery), specifications (e.g., XML Schema), update languages (e.g., [25]), subscription systems (e.g., [5]) and XML access control (e.g., [8]). There is thus a need to study fundamental properties of the XPath language, and in particular to investigate static analyses of XPath queries.

The most basic static analysis of a query language is *satisfiability*: given a query in the language, does there exist a document (or database) on which it returns a nonempty answer? Satisfiability analysis of XPath is important for XML query and update optimization. Consider, for instance, an XML query construct commonly used: "*for* $\$x$ *in* p $c(\$x)$", where p is an XPath expression and $c(\$x)$ is a query or update. If one can decide, at compile time, that p is not satisfiable, then the unnecessary computation of $c(\$x)$ can be simply avoided. Furthermore, XPath satisfiability is critical as a foundation for other consistency problems, such as information leakage in security views [8], type-checking of transformations [14], and consistency of XML specifications.

PODS 2005 June 13-15, 2005, Baltimore, Maryland.

For relational languages, satisfiability analysis is fairly trivial for *positive queries* such as (union of) conjunctive queries and positive fragments of Datalog, while it is trivially undecidable for the most prominent query languages with negation, such as relational calculus and stratified Datalog. For this reason, static analysis for relational languages has focused on the containment problem. In contrast, XPath satisfiability analysis is neither trivial nor futile. A variety of factors contribute to its complexity, such as the operators allowed in XPath queries, combinations of these operators, as well as their interaction with a schema.

Analysis of XPath satisfiability. We now examine factors which XPath satisfiability analysis should take into account.

XPath Fragments. The XPath 1.0 standard [6] contains a large array of operators. We consider several dichotomies, focusing on operators commonly found in practice.

- **positive** vs. **non-positive**: XPath allows qualifiers to be built up with a general negation operator, enabling it to express non-monotone queries; queries without negation, referred to as *positive queries*, can be expressed in existential logic, while queries with negation may have existential and universal quantifiers;
- **downward** vs. **upward**: some XPath queries specify downward traversal (child, descendant), while others also have upward modalities (parent, ancestor);
- **recursive** vs. **non-recursive**: some queries express navigation along the descendant (resp. ancestor) axis, while others only use the child (resp. parent) axis;
- **qualified** vs. **non-qualified**: queries may or may not contain qualifiers (predicates testing properties defined in terms of other queries);
- **with** vs. **without data values**: queries may or may not contain comparisons of data values reached via different navigation paths, expressing joins.

Even positive XPath queries defined in terms of simple operators may be unsatisfiable.

Furthermore, as will be seen later, with different combinations of these operators, the complexity of satisfiability analysis ranges from PTIME to undecidable. From a practical point of view, many applications typically use only a limited set of operators. For example, XML Schema specifies integrity constraints with an XPath fragment that does not support upward modalities. This motivates the study of satisfiability for various XPath fragments (i.e., different combinations of operators).

The impact of schema. XML documents often come with a schema, typically a DTD. Any practical static analysis must take the schema into account. A DTD is far more complex than a relational schema, and it imposes structural constraints such as co-existence of certain siblings (by means of concatenation in the regular expressions within a DTD), exclusive relations on siblings (disjunction), as well as a limited form of negation (excluding certain children). These constraints interact with XPath queries in an intricate way. Indeed, as will be seen later, satisfiability analysis is significantly impacted by the presence or absence of recursion (cycles) and disjunction in a DTD. The role of a schema gives the XPath satisfiability problem an additional dimension.

XPath vs. other query formalisms. For relational query languages with negation, e.g., SQL, the satisfiability problem is typically undecidable. For XPath the picture is far from clear, even in the presence of data values. While XPath is a rich language, it operates on documents whose underlying navigational structure is restricted to be a tree. Furthermore, XPath is a modal language, which, unlike relational calculus and the tree patterns of [1], does not allow the explicit use of free variables. As a result XPath queries can "see" only one node at a time when navigating a tree. This restriction becomes more prominent in the presence of data values or negation, since it allows only a restricted form of data joins to be expressed. This makes the expressive power of XPath quite distinct from other XML query languages, particularly from XML query algebras [12, 23]. The limited expressiveness significantly decreases the complexity of satisfiability analysis in the presence of negation. We will show that in the absence of data values the satisfiability problem for XPath fragments with negation is in EXPTIME, and if recursive axes are further disallowed, it is in PSPACE. This contrasts with first-order logic over trees, the most natural example of a tree query language with explicit variables: there the satisfiability problem has non-elementary complexity [24], even when only the child relation is present.

Taken together, these factors lead to a rich spectrum of languages and satisfiability problems with features quite distinct from containment analysis and relational satisfiability.

Main results. Here we present a comprehensive picture of the satisfiability problem for a variety of XPath fragments in the presence and in the absence of DTDs.

Positive XPath. We begin with a minimal XPath fragment with only the child axis in the presence of DTDs. We then investigate the impact of adding qualifiers, union, upward traversal (the parent axis), recursion (the descendant and ancestor axes) and data values, one at a time, establishing the complexity of the satisfiability problem for each of these fragments. We show that the complexity here ranges from PTIME to NP-complete (Section 4).

XPath with negation. We then investigate a minimal XPath fragment with negation. Since negation makes sense only in the presence of qualifiers, we begin with the XPath fragment with qualifiers, negation and the child axis. We show that the satisfiability problem for this fragment is PSPACE-complete in the presence of DTDs. We then look at the impact of adding upward modality and recursion, one at a time. The complexity here will vary between PSPACE-complete and EXPTIME-complete. Finally, we show that the combination of data values and negation makes a big difference, by adding data values to these fragments. We find that the complexity is in NEXPTIME in the absence of recursive and upward axes, and is undecidable in the general case (Section 5).

Particular DTDs. To explore the impact of different DTD constructs on XPath satisfiability analysis and to understand the interaction between XPath queries and DTD constructs, we also investigate XPath satisfiability under various restricted DTDs. More specifically, we consider non-recursive DTDs, fixed DTDs, and disjunction-free DTDs (i.e., the regular expressions in a DTD do not contain disjunction; we do not consider other kinds of restricted DTDs like star-free, 1-unambiguous or order-independent DTDs). We show that the worst-case complexity does not diminish when the DTD is fixed, but can decrease dramatically in the absence of DTD recursion or disjunction. Finally, we revisit the satisfiability problem for all these fragments *in the absence of DTDs*. We show that for positive XPath, the absence of DTDs simplifies satisfiability analysis, but this is not the case for those fragments with negation (Section 6).

Reductions between problems. We also provide basic results for the connections between XPath satisfiability and XPath containment, between XPath satisfiability analysis in the presence of DTDs and that in the absence of DTDs, and between XPath satisfiability under arbitrary DTDs and under a simple normal form of DTDs. These give us the first results for the containment problem for XPath fragments with the general XPath negation operator, show that XPath satisfiability in the absence of DTDs can be reduced to XPath satisfiability in the presence of DTDs, and allow us to simplify our proofs when handling DTDs (Section 3).

We establish matching upper and lower bounds in all the cases without data values, and several complexity results for fragments with data values. To our knowledge, this work is the first detailed theoretical study of XPath satisfiability in the presence of DTDs, under restricted DTDs, and in the absence of DTDs. A variety of techniques are used to prove these results, including alternating tree automata, rewriting systems, finite-model theoretic constructions, bounded-branching results, and a wide range of reductions.

Brief comparison with prior work. Static analysis of XML queries has for the most part been developed along the lines laid down in the relational theory. In the relational setting, the emphasis has been on the containment problem (given Q_1 and Q_2, does Q_1 always return a subset of Q_2) for positive queries. There is limited practical or theoretical motivation for satisfiability study in the relational case: a user can never propose a conjunctive query that is "nonsensical", so there is no pressing need to check this; and for more general relational queries, a complete satisfiability checking is theoretically impossible. In analogy with this, prior work on XPath static analysis has concentrated on the containment problem for positive query fragments [7, 16, 20, 28]. These are the analogs of positive queries for tree-like data, asserting the existence of a certain kind of tree as a substructure of the document. Since many of these formalisms allow constraints among multiple nodes or the explicit use of variables, their expressiveness and succinctness compared to XPath 1.0 is not clear. While the satisfiability problem is subsumed by the complement of the containment problem for XPath, we will see that the upper bounds on satisfiability derived from previous work on containment for positive XPath fragments are far from tight. Moreover, for fragments with the XPath negation operator, upward modalities and data-value joins, the containment problem has not been studied, with or without DTDs. Thus previous results for XPath containment shed little light on XPath satisfiability analysis.

As we have remarked, XPath satisfiability is both theoretically interesting and practically important, in contrast to its relational counterpart. However, to our knowledge, the satisfiability problem has only been studied for (positive) tree-patterns [13] in the presence of (restricted) DTDs, and for certain XPath fragments in the absence of DTDs [10]. A major exception is [15] which proves EXPTIME upper and lower bounds on XPath with negation and all axes, but without data values, in the presence of DTDs. The results of [15] are discussed in Section 5. The satisfiability problem for a number of practical and interesting fragments, especially those with negation and data values, has not been studied, with or without DTDs. A more detailed overview of related work can be found in Section 7.

Organization. Section 2 reviews DTDs and XPath fragments. Sections 3, 4, 5 and 6 establish technical results as outlined above. Section 7 summarizes our main results and discusses related work. We refer for the proofs to the full version of the paper.

2. NOTATIONS: DTDS AND XPATH FRAGMENTS

In this section, we first review DTDs [4] and then define the fragments of XPath [6] studied in this paper.

2.1 DTDs

Without loss of generality, we represent a Document Type Definition (DTD [4]) 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. We do not consider additional DTD features such as default values and attribute domains.

An XML document is typically modeled as a (finite) node-labeled tree [4], 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* (*subelements*) such that their labels are in the regular language defined by $P(A)$; and (4) for each $a \in R(A)$, each A element n has a unique *a attribute value* which we denote by $n.a$. We call T an *XML tree* of D if $T \models D$.

We also study the following special forms of DTDs.

A *normalized DTD* is a DTD in which for each A in Ele, $P(A)$ is of the following form:

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

where ϵ is the empty word, B_i is a type in Ele (referred to as a *child type* of A), and '+', ',' and '$*$' denote *disjunction*, *concatenation* and the Kleene star, respectively (here we use '+' instead of '|' to avoid confusion). We will see later (Proposition 3.4) that there is often no loss of generality in restricting to normalized DTDs.

A DTD D is said to be *disjunction-free* if for any element type $A \in Ele$, $P(A)$ does not contain disjunction '+'.

A DTD D is *recursive* if the dependency graph of D (which contains an edge (A, B) iff B is in $P(A)$) has a cycle.

A recursive 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 A that satisfies D. One can determine whether an element type A in D is terminating or not in $O(|D|)$ time, where $|D|$ is the size of D; indeed, it can be reduced to the emptiness problem for a CFG, which can be determined in linear time (cf. [11]). Thus to simplify the discussion, in the sequel we assume that all element types in a DTD are terminating. All the complexity results (lower bounds and upper bounds) in this paper remain unchanged in the presence of non-terminating element types.

Example 2.1: Consider an instance $\phi = C_1 \wedge \cdots \wedge C_n$ of 3SAT (cf. [22]), where $C_i = l_{(i,1)} \vee l_{(i,2)} \vee l_{(i,3)}$, and $l_{i,j}$ is a literal of the form x_s or $\bar{x}_s$, and x_s is a propositional variable. Assume that the variables in ϕ are $x_1, \ldots, x_k$. Given ϕ, we define a DTD $D_\phi = (Ele, Att, P, R, r)$, where $Ele = \{r, T, F, X_1, \ldots, X_k\}$, $Att = \emptyset$, and P, R are as follows:

P: $r \to X_1, \ldots, X_k$, $\quad X_i \to T + F$, for $i \in [1, k]$;
$\quad\quad T \to \epsilon$, $\quad\quad F \to \epsilon$;
R: $R(A) = \emptyset$, for any $A \in Ele$

An XML tree of D_ϕ lists all the variables X_i under the root, and gives a truth value (T or F) under each X_i. The DTD is normalized and non-recursive; it is not disjunction free. □

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. We define the largest class of XPath queries considered in this paper, referred to as $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg)$, syntactically as follows:

$$p \quad ::= \quad \epsilon \mid l \mid \downarrow \mid \downarrow^* \mid \uparrow \mid \uparrow^* \mid p/p \mid p \cup p \mid p[q],$$

where ϵ and l 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; '/' and '$\cup$' denote concatenation and union, respectively; and finally, q in $p[q]$ is called a *qualifier* and is defined by:

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

where p, p' are as defined above, A is a label in Σ, op is either '$=$' or '$\neq$', 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.

A query p in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg)$ 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 nodes 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 $p = p_1/p_2$, then there exists a node v in T such that $T \models p_1(n, v) \wedge p_2(v, n')$;
8. if $p = p_1 \cup p_2$, then $T \models p_1(n, n') \vee p_2(n, n')$;
9. if $p = p_1[q]$, then $T \models p_1(n, n')$ and $T \models q(n')$, where q is a unary predicate of the following cases:
   (a) q is p_2: then $T \models \exists n''\ p_2(n', n'')$;
   (b) q is $lab() = A$: then the label of n' is A;
   (c) q is $p_2/@a \ \mathsf{op}\ `c'$: then $T \models \exists n_1\ (p_2(n', n_1) \wedge n_1.a \ \mathsf{op}\ `c')$, where $n_1.a$ denotes the value of the a attribute of n_1; that is, there exists a node n_1 in T such that $T \models p_2(n', n_1)$, n_1 has attribute a and $n_1.a \ \mathsf{op}\ `c'$;
   (d) q is $p_2/@a \ \mathsf{op}\ p_2'/@b$: then T satisfies the existential formula: $T \models \exists n_1\ \exists n_2\ (p_2(n', n_1) \wedge p_2'(n', n_2) \wedge n_1.a \ \mathsf{op}\ n_2.b)$;
   (e) q is $q_1 \wedge q_2$: then $T \models (q_1(n') \wedge q_2(n'))$;
   (f) q is $q_1 \vee q_2$: then $T \models (q_1(n') \vee q_2(n'))$;
   (g) q is $\neg q'$: then $T \not\models q'(n')$; for instance, if q is $\neg p_2$, then $T \models \forall n''\ \neg p_2(n', n'')$.

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

We 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, $r[\![p]\!] \neq \emptyset$, i.e., the set of nodes reachable from the root of T via p is nonempty. Similarly, we talk about T satisfying a qualifier q if $T \models q(r)$. To simplify the discussion, 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 also investigate various fragments of the language $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg)$. We denote a fragment $\mathcal{X}$ by listing the operators supported by $\mathcal{X}$: the presence or absence of negation '$\neg$', data values '$=$', upward traversal '$\uparrow$' ('$\uparrow^*$'), recursive axis '$\downarrow^*$' ('$\uparrow^*$'), qualifiers '[ ]', wildcard '$\downarrow$', and union and disjunction '$\cup$' (the absence of '$\cup$' indicates that *neither* union *nor* disjunction is allowed). The concatenation operator '/' is included in all the fragments by default. For example, a small fragment with negation is $\mathcal{X}(\downarrow, [\,], \neg)$ (note that '$\neg$' can only appear in qualifiers), and the largest positive fragment is $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =)$.

All these fragments have been found useful in practice. For example, $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ is used by XML Schema to specify integrity constraints, and $\mathcal{X}(\downarrow, \downarrow^*, [\,])$ is the class of tree-pattern queries studied in [1, 21, 27].

Example 2.2: Recall the 3SAT instance ϕ and the DTD D_ϕ given in Example 2.1. One can use a query p in $\mathcal{X}(\cup, [\,])$ to encode ϕ, where p is specified as:

$$\begin{aligned} p &= \epsilon[q_1 \wedge \cdots \wedge q_n] \\ q_i &= \mathsf{XP}(l_{i,1}) \vee \mathsf{XP}(l_{i,2}) \vee \mathsf{XP}(l_{i,3}) \end{aligned}$$

where $\mathsf{XP}(l_{j,j}) = X_s/T$ if $l_{i,j}$ is x_s, and $\mathsf{XP}(l_{j,j}) = X_s/F$ if $l_{i,j}$ is $\bar{x}_s$.

Indeed, ϕ is satisfiable iff there is an XML tree T of the DTD D_ϕ such that $T \models p$. □

Example 2.3: Consider the DTD $D = (Ele, Att, P, R, r)$, where $Ele = \{r, A\}$, $Att = \emptyset$, and P, R are as follows:

$$\begin{aligned} &P: \quad r \to A^*; \\ &R: \quad R(r) = R(A) = \emptyset. \end{aligned}$$

Consider the XPath query $p = B$. Then it is clear that there exists no XML tree T of the DTD D such that $T \models p$. □

3. THE SATISFIABILITY PROBLEM

We are interested in the satisfiability problem for XPath queries considered together with a DTD: that is, 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 $T \models (p, D)$?

Below we present several basic results for $\mathsf{SAT}(\mathcal{X})$.

3.1 Satisfiability in the Absence of DTDs

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 $T \models p$.

This version of the satisfiability problem for $\mathcal{X}$ is actually a special case of $\mathsf{SAT}(\mathcal{X})$, since it can be reduced to $\mathsf{SAT}(\mathcal{X})$ when the input DTD is fixed to range over DTDs of the form $D_p = (Ele_p, Att_p, P_p, R_p, r_p)$, where (1) Ele_p consists of a distinct label X as well as all the labels A mentioned in p in the form of a sub-query A or a qualifier $\mathsf{lab}() = A$; (2) Att_p consists of all the attribute names a, b mentioned in p in the form of a qualifier $p/@a$ op 'c' or $p/@a$ op $p'/@b$; (3) for each $A \in Ele_p$, the production for A is defined to be $A \to (A_1 + \cdots + A_n)^*$, where $Ele_p = \{A_1, \ldots, A_n\}$; (4) $R_p(A)$ is defined to be Att_p; and (5) r_p is one of A_i's in Ele_p. The connection between this satisfiability problem and $\mathsf{SAT}(\mathcal{X})$ is encapsulated in the following.

Proposition 3.1: *For any fragment $\mathcal{X}$ of XPath queries defined above and any query p in $\mathcal{X}$, there exists an XML tree T such that $T \models p$ iff there exists an XML tree T' such that $T' \models (p, D)$, where D has the form of D_p given above.* □

For a query p, there are at most $O(|p|)$ many such DTDs D (by allowing r_p to range over all the element types in Ele_p), and the size of such D is in $O(|p|^2)$. As a result, since all the upper bounds for $\mathsf{SAT}(\mathcal{X})$ established in this paper are with respect to complexity classes containing PTIME, they also hold for the satisfiability problem for $\mathcal{X}$ in the absence of DTDs. However, we shall see that for some fragments $\mathcal{X}$, the complexity for its satisfiability problem in the absence of DTDs can be much lower than its counterpart for the same fragment $\mathcal{X}$ in the presence of DTDs.

3.2 XPath Satisfiability and Containment

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 $p_1, p_2 \in \mathcal{X}$ and a DTD D, whether or not for any XML tree T of D, $r[\![p_1]\!] \subseteq r[\![p_2]\!]$, where r is the root of T. That is, whether the answer to p_1 is contained in the answer to p_2 over all the XML trees of D. If this holds then we say that $p_1 \subseteq p_2$ *under* D.

For any fragment $\mathcal{X}$, $\mathsf{SAT}(X)$ is reducible to the complement of $\mathsf{CNT}(\mathcal{X})$. Indeed, given any query $p \in X$ and DTD D, (p, D) is satisfiable iff $p_1 \not\subseteq \emptyset_D$, where $\emptyset_D$ is a special query that returns an empty set over any XML tree of D. Note that $\emptyset_D$ is definable in any of our $\mathcal{X}$'s (e.g., $\emptyset_D$ can be defined to be A where A is not an element type of D). Recall that for a complexity class K, co$\mathsf{K}$ stands for $\{\bar{P} \mid P \in \mathsf{K}\}$. So we have observed:

Proposition 3.2: *For any class $\mathcal{X}$ of XPath queries defined above, 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.* □

We shall see that the upper bound for $\mathsf{SAT}(\mathcal{X})$ is often much lower than its counterpart for $\mathsf{CNT}(\mathcal{X})$. Furthermore, for many fragments $\mathcal{X}$ considered in this paper, (e.g. the fragments $\mathcal{X}(\ldots, \neg)$ with negation) the complexity of $\mathsf{CNT}(\mathcal{X})$ has not been established by previous work.

For fragments $\mathcal{X}$ supporting certain operators, $\mathsf{SAT}(\mathcal{X})$ and the complement problem of $\mathsf{CNT}(\mathcal{X})$ actually coincide. Consider the following two cases.

- 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;
- Any class containing negation and closed under the *inverse operator* defined by $\mathsf{inverse}(\downarrow) = \uparrow$ and $\mathsf{inverse}(\downarrow^*) = \uparrow^*$.

Proposition 3.3: *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 $\mathsf{SAT}(\mathcal{X})$.* □

In Section 5, we will apply Propositions 3.2 and 3.3 to get complexity results for $\mathsf{CNT}(\mathcal{X})$ based on the results for $\mathsf{SAT}(\mathcal{X})$ established later on. To the best of our knowledge, our complexity results for $\mathsf{CNT}(\mathcal{X})$ are the first results for the containment problem for the fragments with negation.

3.3 XPath Satisfiability and Normalized DTDs

A mild variant of $\mathsf{SAT}(\mathcal{X})$ for a fragment $\mathcal{X}$ is the problem to determine, given any query $p \in \mathcal{X}$ and any *normalized DTD* D, whether or not there is an XML tree T such that $T \models (p, D)$. Let us refer to this problem as *the satisfiability problem for $\mathcal{X}$ under normalized DTDs*. The next result tells us that for many fragments $\mathcal{X}$, $\mathsf{SAT}(\mathcal{X})$ and satisfiability for $\mathcal{X}$ under normalized DTDs are polynomially equivalent, i.e., there are PTIME reductions in both directions.

Proposition 3.4: *For any class $\mathcal{X}$ of XPath queries that allows '$\cup$' and '$\downarrow$' (and in addition, label test $\mathsf{lab}() = A$ if $\mathcal{X}$ allows upward modalities), there exists a linear-time function N from DTDs to normalized DTDs, and there exists a PTIME computable function $f : \mathcal{X} \to \mathcal{X}$ such that, for any DTD D and any XPath query $p \in \mathcal{X}$, there exists an XML*

tree T such that $T \models (p, D)$ iff there exists an XML tree T' such that $T' \models (f(p), N(D))$. Moreover, $N(D)$ does not introduce DTD constructs ('$,+,*$') not already in D. □

Since the satisfiability problem for $\mathcal{X}$ under normalized DTDs is a special case of $\mathsf{SAT}(\mathcal{X})$, this proposition says that it suffices to consider normalized DTDs when proving either upper or lower bounds for fragments satisfying the restriction above; we will make use of this frequently in our proofs.

4. POSITIVE XPATH QUERIES

In this section we study satisfiability of XPath queries without negation, namely, queries in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =)$, referred to as *positive XPath queries*. We investigate $\mathsf{SAT}(\mathcal{X})$ for various sub-classes $\mathcal{X}$ of this fragment.

As observed in [2], positive XPath queries in the fragment $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,])$ can be expressed in a fragment of positive existential first-order logic ($\exists^+\mathsf{FO}$) over trees, built up from unary label predicates, binary predicates *child* and *descendant*, and closed under $\wedge, \vee$ and $\exists$. A mild (two-sorted) extension of the fragment $\exists^+\mathsf{FO}$ can express queries in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =)$, by supporting unary attribute function, and equality and inequality on attribute values. This fragment does not use universal quantifiers ($\forall$).

Given this existential characterization, it is not surprising that the satisfiability problem is in NP. However, we will find that for even limited positive languages it is NP-hard. We start with a small fragment $\mathcal{X}(\downarrow, \downarrow^*, \cup)$, i.e., the downward fragment without qualifiers and data values. We then investigate the impacts of different operators on the complexity of satisfiability analysis of positive XPath queries, extending $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ by adding qualifiers, upward traversal, recursive axes and data values.

Downward XPath queries without qualifiers. We first consider $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ queries. There exists an algorithm (based on dynamic programming) that, given any DTD D and any query p in $\mathcal{X}(\downarrow, \downarrow^*, \cup)$, decides whether or not (p, D) is satisfiable in $O(|p| \times |D|^2)$ time, where $|p|$ and $|D|$ denote the sizes of p and D, respectively. Thus we have:

Theorem 4.1: $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup))$ *is in PTIME.* □

Recall that the containment problem $\mathsf{CNT}(\mathcal{X}(\downarrow, \downarrow^*, \cup))$ is EXPTIME-complete [20]. This shows that satisfiability analysis is quite different from its containment counterpart.

In contrast to Theorem 4.1, the satisfiability problem for $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ becomes trivial in the absence of DTDs.

Proposition 4.2: *In the absence of DTDs, all queries in $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ are always satisfiable.* □

Obviously, this is an extreme case where the presence of DTDs complicates satisfiability analysis. This result is a special case of Theorem 6.10, which, along with the remaining complexity results for the satisfiability problem in the absence of DTDs, will be presented in Section 6.

Downward XPath queries without qualifiers. We now study $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$, i.e., the extension of $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ by adding qualifiers. The result below shows that adding qualifiers does make our lives harder: the satisfiability problem becomes intractable, even in the absence of recursion ($\downarrow^*$), and without either disjunction (union $\cup$) or wildcard ($\downarrow$).

Proposition 4.3: *The following problems are NP-hard:*

1. $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,]))$;
2. $\mathsf{SAT}(\mathcal{X}(\cup, [\,]))$.

□

PROOF SKETCH. These are verified by reduction from 3SAT, which is NP-complete (cf. [22]). A reduction from 3SAT to $\mathsf{SAT}(\mathcal{X}(\cup, [\,]))$ has been given in Examples 2.1 and 2.2. The reduction to $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,]))$ can be found in the full version of the paper. □

Upward XPath queries without qualifiers. Alternatively we extend $\mathcal{X}(\downarrow, \downarrow^*, \cup)$ by allowing upward modalities. The presence of upward modalities also complicates satisfiability analysis: the satisfiability problem also becomes intractable, even in the absence of recursion ($\downarrow^*, \uparrow^*$), union ($\cup$) and qualifiers ($[\,]$). This is shown by the result below, which can also be proved by reduction from 3SAT.

Proposition 4.4: $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$ *is NP-hard.* □

Adding recursion and data values. For positive XPath queries with qualifiers, the presence of recursion and data values does not increase the complexity of satisfiability analysis. Indeed, below we show that adding recursion and data values does not move the problem beyond NP.

Theorem 4.5: $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =))$ *is in NP. Furthermore,* $\mathsf{SAT}(\mathcal{X})$ *is NP-complete for any fragment $\mathcal{X}$ of $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =)$ that supports either $(\downarrow, [\,])$, or $(\cup, [\,])$, or $(\downarrow, \uparrow)$.* □

PROOF SKETCH. The upper bound is proved by providing a NP algorithm for checking the satisfiability of queries in $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =)$ in the presence of DTDs. For a given a query p and DTD D, the algorithm first computes a so-called skeleton T_p of p and then guesses an embedding of T_p (or witness skeleton) in a tree using D, without explicitly constructing the tree. The main result is that it is sufficient to guess a witness skeleton of bounded depth. More specifically, a witness skeleton can be obtained by guessing at most $|p|$ paths in D of length bounded by $(|p| + |p|^2) \times |D|$, and then verifying whether these paths can expand to a tree conform with D. This can be done in time $O(|\text{witness skeleton}| \times |D|)$. The NP-completeness follows from this and Propositions 4.4 and 4.3. □

In contrast, it has been shown in [20, 28] that in the presence of DTDs, the containment problem $\mathsf{CNT}(\mathcal{X})$ is EXPTIME-hard in the presence of DTDs, when $\mathcal{X}$ is either $\mathcal{X}(\downarrow^*, \cup)$

or $\mathcal{X}(\downarrow, \downarrow^*, [\,])$; and it is in EXPTIME for $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$. Neither XPath's data value equality nor upward modalities are considered in [20, 28].

5. XPATH FRAGMENTS WITH NEGATION

In this section we show that allowing negation in qualifiers makes satisfiability analysis different in nature. With negation one must deal with both universal and existential quantifiers. In contrast to positive XPath queries, adding data values and/or recursion to XPath fragments with negation has an enormous effect: with recursive axes, it makes the satisfiability problem undecidable, while without recursion there is a jump to NEXPTIME. That is, the interaction between recursion, data values and negation is rather intricate.

Most previous work on XPath containment/satisfiability bounds [7, 10, 13, 16, 20, 28] has considered either no negation or restricted negation. From the EXPTIME lower bounds on containment in [20, 28] plus Proposition 3.3, we get an EXPTIME lower bound for $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], \neg))$. [15] considers an extension of XPath which includes $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], \neg)$, and proves an EXPTIME bound via reduction to dynamic logic. We include this result here for completeness.

We first study the impact of negation by investigating $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$. We then gradually extend $\mathcal{X}(\downarrow, [\,], \neg)$ by adding upward modalities, recursion, and data values, investigating the impact of these operators on satisfiability analysis in the presence of negation.

Non-recursive XPath queries with negative qualifiers. We first consider fragments of XPath with negation but without recursion. The result below tells us three things. First, the presence of negation makes satisfiability analysis PSPACE-hard. Second, the bound is tight. Third, in contrast to Propositions 4.3 and 4.4 for positive XPath queries, further extending $\mathcal{X}(\downarrow, [\,], \neg)$ by allowing union ($\cup$) and upward modality ($\uparrow$) does not make the analysis harder. The upper-bound proofs involve radically different techniques from those used in either the NP membership in the previous section, or the EXPTIME bounds of [20].

Theorem 5.1: $\mathsf{SAT}(\mathcal{X})$ *is PSPACE-complete for any fragment* $\mathcal{X}$ *that (1) contains* $\mathcal{X}(\downarrow, [\,], \neg)$*, and (2) is contained in* $\mathcal{X}(\downarrow, \uparrow, \cup, [\,], \neg)$. □

PROOF SKETCH. For the lower bound, we show that $\mathsf{SAT}(\mathcal{X}(\downarrow, [\ ], \neg))$ is PSPACE-hard by reduction from 3QSAT, a well-known PSPACE-complete problem (cf. [22]). For the upper bound, we show that $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \cup, [\,], \neg))$ is in PSPACE by first establishing a linear bound on the maximum branching needed in a witness model. Using this we then provide a NPSPACE algorithm for checking the satisfiability of queries in $\mathcal{X}(\downarrow, \uparrow, \cup, [\,], \neg)$ in the presence of DTDs (recall that PSPACE = NPSPACE).

More specifically, for a given query p and DTD D, the algorithm first computes an ordered list Q of all the sub-queries in p. Next, the algorithm is implemented by using a function $\mathsf{check}(n, s)$, which (a) takes a node n labeled A and a natural number s as input, (b) guesses a subtree rooted at n that conforms to the DTD D and has a depth bounded by $|p| - s$, without explicitly constructing the subtree (here, we use the bound on the maximum branching needed), and (c) evaluates all the queries q in Q at all the nodes in the subtree; it returns for each $q \in Q$ a Boolean formula, denoted by $\mathsf{sat}(q, n)$, which is either true if q is satisfied at n in the subtree, false if q is not satisfied at n in the subtree, or which tells what should be evaluated at the parent of n in order to establish the truth value (in the presence of $\uparrow$). The size of $\mathsf{sat}(q, n)$ is bounded by $|p|$.

The computation can be done in NPSPACE although it takes exponential time in $|D|$. The algorithm does not explicitly construct the XML tree, which is of possibly exponential size; instead, at any time it only constructs a partial subtree of size $O(k \times |p|)$. Here k is the bound on the maximum branching needed. At each of its node a list of size $O(|p|^2)$ is used, bringing the total space complexity to $O(k \times |p|^3)$. □

Recursive XPath queries with negative qualifiers. We now study the impact of recursion ($\downarrow^*, \uparrow^*$) on satisfiability analysis. Recall from Theorem 4.5 that the presence of recursion does not make the analysis harder for positive XPath queries. In contrast, Theorem 7 ii) of [15] implies that the addition of recursion to $\mathcal{X}(\downarrow, [\,], \neg)$ makes the problem EXPTIME-hard. The upper bound in the same theorem implies that the bound above is tight, even when upward traversal ($\uparrow, \uparrow^*$) and union ($\cup$) are allowed (indeed, [15] shows that this upper bound holds even in the presence of specialized DTDs and sibling axes in queries).

Theorem 5.2: *[15]* $\mathsf{SAT}(\mathcal{X})$ *is EXPTIME-complete for any fragment* $\mathcal{X}$ *that (1) contains* $\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg)$*, and (2) is contained in* $\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], \neg)$. □

Adding data values. In contrast to Theorem 4.5, which shows that the presence of data values does not complicate satisfiability analysis of positive XPath queries, we next show that adding data values to fragments with negation has an enormous impact on the analysis.

Theorem 5.3: $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \cup, [\,], =, \neg))$ *is undecidable.* □

PROOF SKETCH. This is verified by reduction from the halting problem for two-register machines, which is known to be undecidable (cf. [3]). □

The good news is that not every fragment with negation and data values is beyond reach: below we show that the satisfiability problem in the presence of data values and negation is still decidable for non-recursive downward queries. Unlike the previous results, the proof uses a finite-model-theoretic construction. While the decidability of the satisfiability problem remains open if the upward axis $\uparrow$ is further added, we show that the "hardness" increases compared to the fragment without data equality.

Theorem 5.4: *(1)* $\mathsf{SAT}(\mathcal{X}(\downarrow,\cup,[\,],=,\neg))$ *is in NEXPTIME, (2)* $\mathsf{SAT}(\mathcal{X}(\uparrow,[\,],=,\neg))$ *is EXPTIME-hard.* □

PROOF SKETCH. We prove the lower bound by reduction from the two-player corridor tiling game, and the upper bound by first establishing a small model property for satisfiable queries in $\mathcal{X}(\downarrow,\uparrow,\cup,[\,],=,\neg)$, and then showing that the existence of such a model can be decided in NEXPTIME.

More specifically, for a given query p and DTD D, we show that if (p, D) has a model (i.e., there exists an XML tree satisfying (p, D)), then it has a small model of exponential size in $|p|$ and $|D|$. The nondeterministic decision algorithm then works as follows: first guess a model T of exponential size in $|p|$ and $|D|$, and then check whether T satisfies (p, D). The latter can be done in polynomial time in $|T|$ and $|p|$ (cf. [9]), and thus the algorithm is in NEXPTIME. Note that it is not necessary to guess and check all possible data values for attributes; the decision algorithm only needs to guess a binary relation '=' between the attribute values and between attribute values and constants mentioned in p. Such a relation has a size bounded by $O(|p|^2)$. □

These results indicate that XPath with negation and data values is very close to the border of decidability, and which side a particular fragment falls on depends on syntactic restrictions on axes and qualifier constructs.

Containment analysis in the presence of negation. The results above, along with Propositions 3.2 and 3.3, give us complexity results for the containment problem $\mathsf{CNT}(\mathcal{X})$ for XPath fragments $\mathcal{X}$ with negation in the presence of DTDs.

Corollary 5.5: *For the containment problem* $\mathsf{CNT}(\mathcal{X})$ *in the presence of DTDs,*

1. $\mathsf{CNT}(\mathcal{X}(\downarrow,[\,],\neg))$ *is PSPACE-hard;*
2. $\mathsf{CNT}(\mathcal{X}(\downarrow,\uparrow,\cup,[\,],\neg))$ *is PSPACE-complete;*
3. $\mathsf{CNT}(\mathcal{X}(\downarrow,\downarrow^*,[\,],\neg))$ *is EXPTIME-hard;*
4. $\mathsf{CNT}(\mathcal{X}(\downarrow,\downarrow^*,\uparrow,\uparrow^*,\cup,[\,],\neg))$ *is EXPTIME-complete;*
5. $\mathsf{CNT}(\mathcal{X}(\downarrow,\uparrow,\cup,[\,],=,\neg))$ *is EXPTIME-hard;*
6. $\mathsf{CNT}(\mathcal{X}(\downarrow,\cup,[\,],=,\neg))$ *is in coNEXPTIME;*
7. $\mathsf{CNT}(\mathcal{X}(\downarrow,\uparrow,\downarrow^*,\uparrow^*,\cup,[\,],=,\neg))$ *is undecidable.*

□

6. SATISFIABILITY ANALYSIS UNDER RESTRICTED DTDS

The hardness results in the previous section leave open the possibility that feasible algorithms exist for restricted DTDs that may occur often in practice. We thus investigate whether or not restricted DTDs simplify the analysis of XPath satisfiability. More specifically, we study $\mathsf{SAT}(\mathcal{X})$ for XPath fragments $\mathcal{X}$ in the following four settings: (1) when DTDs are non-recursive; (2) when DTDs are fixed; (3) when DTDs are disjunction-free; and (4) in the absence of DTDs, which, as shown by Proposition 3.1, is reducible to a special case of $\mathsf{SAT}(\mathcal{X})$. We show that for some restricted DTDs and some fragments $\mathcal{X}$, $\mathsf{SAT}(\mathcal{X})$ has lower complexity. Note that the upper bounds for $\mathsf{SAT}(\mathcal{X})$ also hold for the restricted analysis: if $\mathsf{SAT}(\mathcal{X})$ is in a complexity class K, then satisfiability analysis is also in K under restricted DTDs.

6.1 Non-recursive DTDs

A non-recursive DTD D has the property that for any XML tree T conforming to D, the depth of T, i.e., the length of the longest path from the root to a leaf of T, is bounded by $|D|$. This simplifies the analysis of XPath queries with recursive axes ($\downarrow^*, \uparrow^*$). Specifically, for any $\mathcal{X}(\downarrow,\downarrow^*,\cup,\dots)$, i.e., a fragment with $\downarrow$, recursion $\downarrow^*$, union $\cup$ and possibly other operators, $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,\cup,\dots))$ and $\mathsf{SAT}(\mathcal{X}(\downarrow,\cup,\dots))$ are quadratic-time equivalent under non-recursive DTDs, where $\mathcal{X}(\downarrow,\cup,\dots)$ denotes the same fragment without $\downarrow^*$. Indeed, there is a quadratic-time reduction from $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,\cup,\dots))$ to $\mathsf{SAT}(\mathcal{X}(\downarrow,\cup,\dots))$. Intuitively, given a non-recursive DTD D, one can eliminate recursion in a query by replacing $\downarrow^*$ with $(\epsilon \cup \downarrow \cup \dots \cup \downarrow^{|D|})$, where $\downarrow^n$ abbreviates the n-fold concatenation of $\downarrow$; similarly for $\mathcal{X}(\uparrow,\uparrow^*,\cup,\dots)$, i.e., a fragment with $\uparrow$, $\uparrow^*$, $\cup$ and other operators. This is stated below.

Proposition 6.1: *Under non-recursive DTDs, the following problems are quadratic-time equivalent:*

1. $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,\cup,\dots))$, $\mathsf{SAT}(\mathcal{X}(\downarrow,\cup,\dots))$;
2. $\mathsf{SAT}(\mathcal{X}(\uparrow,\uparrow^*,\cup,\dots))$, $\mathsf{SAT}(\mathcal{X}(\uparrow,\cup,\dots))$;
3. $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,\uparrow,\uparrow^*,\cup,\dots))$, $\mathsf{SAT}(\mathcal{X}(\downarrow,\uparrow,\cup,\dots))$;

where $\mathcal{X}(\downarrow,\uparrow,\cup,\dots)$ *supports the same set of operators as* $\mathcal{X}(\downarrow,\downarrow^*,\uparrow,\uparrow^*,\cup,\dots)$ *except* $\downarrow^*,\uparrow^*$*; similarly for* $\mathcal{X}(\downarrow,\cup,\dots)$ *and* $\mathcal{X}(\uparrow,\cup,\dots)$. □

From the proposition it follows that the EXPTIME problem of Theorem 5.2 collapses to PSPACE (Theorem 5.1), and that $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,\cup,[\,],=,\neg))$ is now known to be decidable (Theorem 5.4) under non-recursive DTDs.

Corollary 6.2: *Under non-recursive DTDs,*

1. $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,\uparrow,\uparrow^*,\cup,[\,],\neg))$ *is in PSPACE, and*
2. $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,\cup,[\,],=,\neg))$ *is in NEXPTIME.*

That is, they are equivalent to $\mathsf{SAT}(\mathcal{X}(\downarrow,\uparrow,\cup,[\,],\neg))$ *and* $\mathsf{SAT}(\mathcal{X}(\downarrow,\cup,[\,],=,\neg))$*, respectively.* □

One might be tempted to think that non-recursive DTDs might also lower the NP, PSPACE, and EXPTIME bounds given earlier. However, the proofs of Propositions 4.3, 4.4, Theorem 5.1 and the lower bound of Theorem 5.4 all utilize non-recursive DTDs. Hence:

Corollary 6.3: *Under non-recursive DTDs,*

1. $\mathsf{SAT}(\mathcal{X}(\downarrow,[\,]))$, $\mathsf{SAT}(\mathcal{X}(\cup,[\,]))$ *and* $\mathsf{SAT}(\mathcal{X}(\downarrow,\uparrow))$ *are NP-hard;*
2. $\mathsf{SAT}(\mathcal{X}(\downarrow,[\,],\neg))$ *is PSPACE-hard.*
3. $\mathsf{SAT}(\mathcal{X}(\uparrow,\cup,[\,],=,\neg))$ *is EXPTIME-hard.*

□

6.2 Fixed DTDs

Under fixed DTDs, the satisfiability problem $\mathsf{SAT}(\mathcal{X})$ is to determine, given any query $p \in \mathcal{X}$, whether or not there exists an XML tree T that satisfies both p and a fixed DTD D_0. Here D_0 is not an input, but is predefined. Together with other restrictions, fixed DTDs may simplify the analysis of $\mathsf{SAT}(\mathcal{X})$. For example, the observation below contrasts with the EXPTIME hardness in Theorem 5.2.

Proposition 6.4: $\mathsf{SAT}(X(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\ ], \neg))$ *is in PTIME under fixed, non-recursive DTDs. Furthermore, if the DTDs do not contain Kleene star,* $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], =, \neg))$ *is in PTIME.* □

Proof sketch. If the fixed DTD D_0 has no Kleene star, then there are only a constant number of tree instances of D_0, all of size bounded by a constant (More specifically, these constants are functions of $|D_0|$). The algorithm now simply evaluates the query p on each instance in PTIME [9] and checks whether any of the query results is non-empty.

If D_0 has a Kleene star, then we show that there exists a (constant) number of non-recursive DTDs D_i, each not using the Kleene star, and such that (p, D) is satisfiable iff (p, D_i) is satisfiable for some D_i. The result now follows from the previous case. □

Unfortunately, fixed DTDs do not make our lives much easier: all the fragments studied in Propositions 4.3 and 4.4 remain intractable under fixed DTDs. These results require more involved encoding arguments than the ones given in Section 4, since the former relied heavily on varying DTDs.

Theorem 6.5: *Under fixed DTDs, the following problems are NP-hard:*

1. $\mathsf{SAT}(\mathcal{X}(\cup, [\,]))$,
2. $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,]))$,
3. $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$.

□

The next result shows that the PSPACE (Theorem 5.1) and the EXPTIME (Theorems 5.2, 5.4) lower bounds remain intact under fixed DTDs; and worse still, so does the undecidability (Theorem 5.3). These results are proven by reduction from 3QSAT, the two player corridor tiling game and the halting problem for two-register machines, respectively.

Theorem 6.6: *Under fixed DTDs,*

1. $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$ *is PSPACE-hard;*
2. $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg))$ *is EXPTIME-hard;*
3. $\mathsf{SAT}(\mathcal{X}(\uparrow, [\,], =, \neg))$ *is EXPTIME-hard;*
4. $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \downarrow^*, \uparrow^*, \cup, [\,], =, \neg))$ *is undecidable.*

□

6.3 Disjunction-Free DTDs

Recall that a DTD is disjunction-free if no productions in it contain disjunction '+' (Section 2). The fact that disjunction-free DTDs are easier to analyze was already noted in other contexts [13, 14]. The absence of disjunction makes satisfiability analysis simpler for certain fragments. Below we show that for a fragment that contains $\mathcal{X}(\downarrow, [\,])$ and $\mathcal{X}(\cup, [\,])$, satisfiability analysis becomes tractable.

Theorem 6.7: *Under disjunction-free DTDs,*

1. $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,]))$ *is in PTIME* ($O(|p| \times |D|^2)$*, where* p, D *are input query and DTD, respectively);*
2. $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow))$ *is in PTIME* ($O(|p| \times |D|^2)$).

□

Proof sketch. For the fragment $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$, the algorithm is based on dynamic programming and is similar to the proof of Theorem 4.1. The reason that we now can allow for qualifiers is that for disjunction-free DTDs, there is no choice in the labels of the children of any context node and hence the truth value of a conjunction of qualifiers is easily determined.

The algorithm for the second fragment $\mathcal{X}(\downarrow, \uparrow)$ first performs a translation from queries in $\mathcal{X}(\downarrow, \uparrow)$ to queries in $\mathcal{X}(\downarrow, [\,])$ and then applies the algorithm of the previous case. □

However, once we extend $\mathcal{X}(\cup, [\,])$ and $\mathcal{X}(\downarrow, [\,])$ by adding data values (=) or upward axes, the disjunction-free distinction no longer affects the worst-case behavior.

Theorem 6.8: *Under disjunction-free DTDs, the following problems are NP-hard:*

1. $\mathsf{SAT}(\mathcal{X}(\cup, [\,], =))$,
2. $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], =))$,
3. $\mathsf{SAT}(\mathcal{X}(\downarrow, \uparrow, \cup, [\,]))$.

The last result holds under fixed, disjunction-free DTDs. □

The absence of disjunction in DTDs also has little impact on fragments with negation: the PSPACE and EXPTIME lower bounds are robust under disjunction-free DTDs. This is because one can encode much of the semantics of disjunction in terms of a combination of the Kleene star in a DTD and XPath qualifiers with negation, as shown in the proof of the next result, which extends the proofs of Theorems 6.6 (for the first two) and 5.4 (for the last one).

Corollary 6.9: *Under disjunction-free DTDs,*

1. $\mathsf{SAT}(\mathcal{X}(\downarrow, [\,], \neg))$ *is PSPACE-hard;*
2. $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, [\,], \neg))$ *is EXPTIME-hard;*
3. $\mathsf{SAT}(\mathcal{X}(\uparrow, [\,], =, \neg))$ *is EXPTIME-hard.*

The first two hold under fixed, disjunction-free DTDs. □

6.4 In the Absence of DTDs

As hinted already in Section 4, the absence of DTDs notably simplifies the analysis of $\mathsf{SAT}(\mathcal{X})$ for certain classes $\mathcal{X}$.

Theorem 6.10: *In the absence of DTDs,*

1. *all queries in* $\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,])$ *are satisfiable if label test (i.e.,* $\mathsf{lab}() = A$*) is disallowed in qualifiers; if label test is allowed,* $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \cup, [\,]))$ *is in PTIME* ($O(|p|^3)$*, where* p *is input query);*

2. $\mathsf{SAT}(\mathcal{X}(\downarrow,\uparrow,[\,]))$ *is in PTIME (*$O(|p^2|)$*);*
3. $\mathsf{SAT}(\mathcal{X}(\downarrow,\uparrow,=))$ *is in PTIME (*$O(|p|^3)$*).*

□

PROOF SKETCH. For the fragment $\mathcal{X}(\downarrow,\downarrow^*,\cup,[\,])$ the algorithm is based on dynamic programming. The algorithm basically checks whether there are conflicting label tests preventing a query to be satisfiable. Although the algorithm is similar to the algorithm in the proof of Theorem 4.1, here we can easily deal with qualifiers in the absence of DTDs. Indeed, because of the existential semantics of $\mathcal{X}(\downarrow,\downarrow^*,\cup,[\,])$ and the absence of DTDs, we can freely generate a separate branch for each qualifier in a conjunction of qualifiers on which the qualifier is satisfied.

For the second fragment $\mathcal{X}(\downarrow,\uparrow,[\,])$, the algorithm normalizes the input query p such that $\uparrow$ does not appear in the scope of $\downarrow$ anymore. After the normalization, we simply check whether the first axis is not $\uparrow$, and if in every qualifier the label equalities are propositional consistent.

Finally, for the fragment $\mathcal{X}(\downarrow,\uparrow,=)$ we rely again on a normalization step. More specifically, we bring any query into the form $\eta_1[q_1]/\ldots/\eta_k[q_k]$, where η_i is either $\downarrow$ or A, $q_i = G_i \wedge H_i$, G_i is a conjunction of conjunctive atoms, H_i is a conjunction of expressions $p_1/@a\ \mathsf{op}\ p_2/@b$, and p_i is in the normal form. Here a conjunctive atom is of the form $\epsilon/@a\ \mathsf{op}\ \epsilon/@b$, $\epsilon/@a\ \mathsf{op}$ 'c' or $\mathsf{lab}() = A$.

Since all label tests and label equalities are neatly organized in a normalized query, we can check its satisfiability by inductively checking the consistency of label tests and label equalities. □

But as with the disjunction-free distinction, in the presence of data value equality (=) or upward modalities ($\uparrow$), the lack of a DTD does not help matters. The proofs of the result below follows from that of Theorem 6.8. Its last part also follows from the results of [10].

Corollary 6.11: *In the absence of DTDs, the following problems are NP-hard:*
1. $\mathsf{SAT}(\mathcal{X}(\cup,[\,],=))$,
2. $\mathsf{SAT}(\mathcal{X}(\downarrow,\uparrow,\cup,[\,]))$.

□

As in the settings of disjunction-free and fixed DTDs, the absence of DTDs does not simplify satisfiability analysis of fragments with negation. This is verified by the results below; the proofs for the first two are mild extensions of their counterparts for disjunction-free DTDs (Corollary 6.9), and the proof for the last one is a variation of that of Theorem 5.4.

Corollary 6.12: *In the absence of DTDs,*
1. $\mathsf{SAT}(\mathcal{X}(\downarrow,[\,],\neg))$ *is PSPACE-hard;*
2. $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,[\,],\neg))$ *is EXPTIME-hard;*
3. $\mathsf{SAT}(\mathcal{X}(\uparrow,[\,],=,\neg))$ *is EXPTIME-hard.*

□

7. CONCLUSION AND RELATED WORK

We have studied the satisfiability problem for a variety of XPath fragments in the presence of DTDs, in the absence of DTDs, and under restricted DTDs.

The main complexity results are summarized in Table 1, annotated with their corresponding theorems.

Under arbitrary (any) DTDs, the table shows that the complexity of $\mathsf{SAT}(\mathcal{X})$ ranges from PTIME to NP-complete when $\mathcal{X}$ is a positive XPath fragment. When negation is added, the complexity ranges from PSPACE-complete to undecidable, depending on different combinations of the negation operator, the recursive axes and data-value joins.

Under non-recursive (non-rec) DTDs, the satisfiability problem becomes much simpler for XPath fragments with recursive axes; however, the absence of DTD recursion does not help satisfiability analysis of XPath fragments without recursive axes. The absence of disjunction (+-free) in DTDs simplifies satisfiability analysis of positive XPath fragments, but does not help fragments with negation. Fixing the DTD has little impact on the worst-case analysis, while the absence of DTDs (DTD-free) diminishes the complexity for positive fragments but not for those fragments with negation.

Our analysis of XPath fragments with negation and data values is still preliminary. An open question is the complexity of $\mathsf{SAT}(\mathcal{X}(\downarrow,\uparrow,\downarrow^*,\uparrow^*,\cup,[\,],=,\neg))$, i.e., the largest fragment with recursive and upward axes, negation and data-value joins, in the absence of DTDs, DTD disjunctions, or DTD recursion. Another open question is the decidability of $\mathsf{SAT}(\mathcal{X}(\downarrow,\downarrow^*,\cup,[\,],=,\neg))$, i.e., the largest downward fragment. We only know that the problem is decidable either under non-recursive DTDs, or in absence of the recursive axis ($\downarrow^*$). Finally, it would also be interesting to see if the NEXPTIME bound for non-recursive DTDs remains valid when queries have access to a document ordering.

Related work. The containment problem has been studied for several XPath fragments in the absence and in the presence of DTDs [7, 16, 20, 28, 15]. Most of the work on containment (except [20, 15]) focuses on *positive* XPath, *without* upward axes and data-value joins. We have shown that the upper bounds inherited from containment are not tight for satisfiability. Similarly, our bounds do not imply anything about containment analysis for those *positive* fragments studied previously. In the presence of *negation*, Propositions 3.3 and 3.2 allow our upper and lower bounds to be carried over to the containment problem for the corresponding fragments.

While [20] proves bounds on containment in the presence of negation and data values (without upward axes), it does not consider the general XPath negation operator, and instead negation is tied to particular equality comparisons. Data values in [20] are introduced in the form of variables. Variables are not an XPath 1.0 notion, and they change the modal nature of the language dramatically. Since this se-

$\downarrow$	$\downarrow^*$	$\uparrow$	$\uparrow^*$	$\cup$	[ ]	=	$\neg$	any DTDs	nonrec. DTDs	fixed DTDs	'+'-free DTDs	DTD-free
+	+			+				PTIME (Th 4.1)	PTIME (Th 4.1)	PTIME (Th 4.1)	PTIME (Th 4.1)	PTIME (Th 3.1, 4.1)
				+	+			NP-complete (Th 4.5)	NP-complete (Th 6.3, 4.5)	NP-complete (Th 6.5, 4.5)	PTIME (Th 6.7)	PTIME (Th 6.10)
+					+			NP-complete (Th 4.5)	NP-complete (Th 6.3, 4.5)	NP-complete (Th 6.5, 4.5)	PTIME (Th 6.7)	PTIME (Th 6.10)
+		+						NP-complete (Th 4.5)	NP-complete (Th 6.3, 4.5)	NP-complete (Th 6.5, 4.5)	PTIME (Th 6.7)	PTIME (Th 6.10)
+	+			+	+			NP-complete (Th 4.5)	NP-complete (Th 6.3, 4.5)	NP-complete (Th 6.5,4.5)	PTIME (Th 6.7)	PTIME (Th 6.10)
+					+	+		NP-complete (Th 4.5)	NP-complete (Th 6.3, 4.5)	NP-complete (Th 6.5, 4.5)	NP-complete (Th 6.8, 4.5)	PTIME (Th 6.10)
				+	+	+		NP-complete (Th 4.5)	NP-complete (Th 6.3, 4.5)	NP-complete (Th 6.5, 4.5)	NP-complete (Th 6.8, 4.5)	NP-complete (Th 6.11, 4.5)
+		+		+	+			NP-complete (Th 4.5)	NP-complete (Th 6.3, 4.5)	NP-complete (Th 6.8, 4.5)	NP-complete (Th 6.8, 4.5)	NP-complete (Th 6.11, 4.5)
+	+	+	+	+	+	+		NP-complete (Th 4.5)	NP-complete (Th 6.3, 4.5)	NP-complete (Th 6.5, 4.5)	NP-complete (Th 6.8, 4.5)	NP-complete (3.1, 6.11, 4.5)
+					+		+	PSPACE-com -plete (Th 5.1)	PSPACE-com -plete (6.2, 6.3)	PSPACE-com -plete (6.6, 5.1)	PSPACE-com -plete (6.9, 5.1)	PSPACE-com -plete (6.12,5.1)
+		+		+	+		+	PSPACE-com -plete (Th 5.1)	PSPACE-com -plete (6.2, 6.3)	PSPACE-com -plete (6.6, 5.1)	PSPACE-com -plete (6.9, 5.1)	PSPACE-com -plete (6.12,5.1)
+	+				+		+	EXPTIME-com -plete (Th 5.2)	PSPACE-com -plete (6.2, 6.3)	EXPTIME-com -plete (6.6, 5.2)	EXPTIME-com -plete (6.9, 5.2)	EXPTIME-com -plete (6.12,5.2)
+	+	+	+	+	+		+	EXPTIME-com -plete (Th 5.2)	PSPACE-com -plete (6.2, 6.3)	EXPTIME-com -plete (6.6, 5.2)	EXPTIME-com -plete (6.9, 5.2)	EXPTIME-com -plete (6.12,5.2)
		+			+	+	+	EXPTIME-hard (Th 5.4)	EXPTIME-hard (Cor 6.3)	EXPTIME-hard (Th 6.6)	EXPTIME-hard (Cor 6.9)	EXPTIME-hard (Cor 6.12)
+				+	+	+	+	NEXPTIME (Th 5.4)	NEXPTIME (Th 5.4)	NEXPTIME (Th 5.4)	NEXPTIME (Th 5.4)	NEXPTIME (Th 3.1, 5.4)
+	+	+	+	+	+	+	+	undecidable (Th 5.3)	?	undecidable (Th 6.6)	?	?

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

mantics of negation and data values is different from ours, our results do not imply the results of [20], and vice versa.

[15] is concerned principally with extensions of XPath, but contains bounds on equivalence and satisfiability for the largest fragments we consider that lack data value equality. [15] proves an EXPTIME upper bound on satisfiability for an extension of XPath, which implies an EXPTIME bound for $\mathsf{SAT}(\mathcal{X}(\downarrow, \downarrow^*, \uparrow, \uparrow^*, \cup, [\,], \neg))$. Indeed, the results of [15] imply an EXPTIME upper bound on the extension of this fragment with the sibling axes (which we do not consider), in the presence of specialized DTDs (roughly speaking, XML Schema), rather than just DTDs. A corresponding EXPTIME hardness result for navigational XPath can be derived from a lower bound on query equivalence of the "XCore" language of [15] (see Section 5).

Closest in spirit to our paper are [10, 13], which are general studies of the satisfiability problem. [10] differs from our work in both the set of operators it considers (e.g., without data values), and in that it assumes the absence of DTDs. It gives PTIME bounds in the presence of qualifiers, sibling axes, upward axes, and a root test. We do not consider sibling axes here, but our results suffice to show that these bounds do not hold in the presence of DTDs. The proofs of [10] also imply that the satisfiability problem is NP-complete when downward axes are supplemented by an intersection operator. The intersection operator is not available in XPath 1.0, so we do not consider it here. Finally, [10] shows that satisfiability is NP-hard in the presence of a complement operator, which is again not supported by XPath. Instead. we consider here the XPath negation operator, proving both lower and upper bounds. Note that our results would also carry over to show that XPath fragments with all of the features of [10] is in EXPTIME.

[13] considers a tree pattern formalism with expressiveness incomparable to XPath. These are tree-shaped, positive queries, with data value equality and inequality along with a node-equality test. Note that node-equality can be used to simulate the intersection operation of [10]. [13] shows that the satisfiability problem is NP-complete for several restrictions of this pattern language in the absence of DTDs. It also investigates the satisfiability of tree pattern queries with limited use of data joins (these can only occur "conjunctively") and node equality and inequality under nonrecursive disjunction-free DTDs. Since these results impose severe syntactic restrictions, all of which make sense only within the particular pattern formalism rather than in XPath, it is difficult to compare the results with ours on positive XPath. [13] does not deal with negation, nor can the XPath negation operator be simulated in the formalism of [13].

Minimization, rewriting and optimization are studied for tree patterns and XPath [1, 9, 21, 27]. Expressiveness of XPath is investigated in [2, 17, 18, 19]. No bounds for satisfiability follow from these works.

There are several powerful logical formalisms on trees for which satisfiability is decidable, principally Monadic Second Order Logic (MSO) [26]. All the XPath fragments we consider that omit data-value equality can be translated into MSO, thus giving a decision procedure. However, MSO on trees (and even first-order logic) has a non-elementary satisfiability problem [24]. Restricted logics such as the μ-calculus do admit EXPTIME decision procedures. While the expressiveness of the μ-calculus is incomparable with XPath, one of our results (Theorem 5.2) relies heavily on the principal technique used for proving μ-calculus decidability. For the fragments with data values we know of no semantics-preserving translation into an existing formalism.

Acknowledgment. We thank Kousha Etessami and Kedar Namjoshi for helpful discussions. We also thank Peter Buneman and Leonid Libkin for their comments. Wenfei Fan is supported in part by EPSRC GR/S63205/01, EPSRC GR/T27433/01 (the Engineering and Physical Sciences Research Council, UK) and NSFC 60228006 (National Science Foundation, China). Floris Geerts is postdoctoral researcher of the FWO Vlaanderen and is supported in part by EPSRC GR/S63205/01.

8. 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 G. M. Kuper. Structural properties of XPath fragments. In *ICDT*, 2003.

[3] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 1997.

[4] 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`.

[5] C. Chan, P. Felber, M. Garofalakis, and R. Rastogu. Efficient filtering of XML documents with XPath expressions. In *ICDE*, 2002.

[6] J. Clark and S. DeRose. *XML Path Language (XPath)*. W3C Recommendation, Nov. 1999.

[7] A. Deutsch and V. Tannen. Containment for classes of XPath expressions under integrity constraints. In *KRDB*, 2001.

[8] W. Fan, C. Chan, and M. Garofalakis. Secure XML querying with security views. In *SIGMOD*, 2004.

[9] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *VLDB*, 2002.

[10] J. Hidders. Satisfiability of XPath expressions. In *DBPL*, 2003.

[11] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation (2nd Edition)*. Addison Wesley, 2000.

[12] H. Jagadish, L. Lakshmanan, D. Srivastava, and K. Thompson. TAX: A tree algebra for XML. In *DBPL*, 2001.

[13] L. Lakshmanan, G. Ramesh, H. Wang, and Z. Zhao. On testing satisfiability of tree pattern queries. In *VLDB*, 2004.

[14] W. Martens and F. Neven. Frontiers of tractablity for typechecking simple XML transformations. In *PODS*, 2004.

[15] M. Marx. XPath with conditional axis relations. In *EDBT*, 2004.

[16] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *JACM*, 51(1):2–45, 2004.

[17] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. In *PODS*, 2001.

[18] M. Murata. Extended path expressions for XML. In *PODS*, 2001.

[19] F. Neven and T. Schwentick. Expressive and efficient languages for tree-structured data. In *PODS*, 2000.

[20] F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *ICDT*, 2003.

[21] D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: Looking forward. In *XMLDM*, 2002.

[22] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[23] S. Paparizos, Y. Wu, L. Lakshmanan, and H. Jagadish. Tree logical classes for efficient evaluation of XQuery. In *SIGMOD*, 2004.

[24] L. Stockmeyer. The complexity of decision problems in automata thoery and logic. Technical report, MIT, 1974.

[25] G. Sur, J. Hammer, and J. Siméon. An XQuery-based language for processing updates in XML. In *PLAN-X*, 2004.

[26] J. Thatcher and J. Wright. Generalized finite autoamta with an application to a decision problem of second-order logic. *Math. Systems Theory*, 2:57–82, 1968.

[27] P. T. Wood. Minimising simple XPath expressions. In *WebDB*, 2001.

[28] P. T. Wood. Containment for XPath fragments under DTD constraints. In *ICDT*, 2003.