# XPath Query Satisfiability is in PTIME for Real-World DTDs

Manizheh Montazerian[1], Peter T. Wood[1], and Seyed R. Mousavi[2]

[1] Birkbeck, University of London, London WC1E 7HX, UK
{gmont05,ptw}@dcs.bbk.ac.uk
[2] Isfahan University of Technology, Isfahan, Iran
srm@cc.iut.ac.ir

**Abstract.** The problem of XPath query satisfiability under DTDs (Document Type Definitions) is to decide, given an XPath query $p$ and a DTD $D$, whether or not there is some document valid with respect to $D$ on which $p$ returns a nonempty result. Recent studies in the literature have shown the problem to be NP-hard or worse for most fragments of XPath. However, in this paper we show that the satisfiability problem is in PTIME for most DTDs used in real-world applications. Firstly, we report on the details of our investigation of real-world DTDs and define two properties that they typically satisfy: being *duplicate-free* and being *covering*. Then we concentrate on the satisfiability problem of XPath queries under such DTDs. We obtain a number of XPath fragments for which the complexity of the satisfiablity problem reduces to PTIME when such real-world DTDs are used.

**Key words:** XPath, Satisfiability, Document Type Definitions

## 1 Introduction

With XML becoming the standard for data exchange, substantial work has been done on XML query processing and optimization [1, 5, 7, 10–12, 14]. Much of the work on query optimization has focused on XPath, since XPath is widely used in XML-related applications to select sets of nodes from an XML document tree. Particularly when an XPath query is to be evaluated over documents known to be valid with respect to a DTD (Document Type Definition), it is possible that the query might be *unsatisfiable*, that is, the query always returns an empty result, no matter what document (valid with respect to the DTD) is queried. Relatively little work has been done on detecting whether a given XPath query is satisfiable [2, 6, 8, 9]. However, it is potentially important to detect unsatisfiable XPath queries and optimize queries to remove expressions that will always return an empty result set. Indeed, Lakshmanan *et al.* show that checking satisfiability as a first step in query processing often yields substantial savings in overall query processing time [9].

XPath supports a wide variety of operators whose presence or absence affects the complexity of the satisfiability problem. This has led to the study of various XPath fragments that include only certain operators. For example, in this paper we study the fragment with child axis (/), descendant axis (//), qualifiers ([ ]), wildcard (*) and union (∪). As in [10], we denote this fragment by $\mathrm{XP}^{\{/,[\,],*,//,\cup\}}$, indicating the operators permitted. Larger fragments allow operators such as negation, additional axes such as parent, ancestor and sibling, as well as comparisons involving data values or node identities.

*Example 1.* The XMark benchmark project[3] is based on an online auction application. A fragment of the XMark DTD is given below:

```
site          (regions, categories, catgraph, people, open_auctions,
               closed_auctions)
categories    (category+)
category      (name, description)
description   (text | parlist)
open_auctions (open_auction*)
open_auction  (initial, reserve?, bidder*, current, privacy?, itemref,
               seller, annotation, quantity, type, interval)
```

with `site` being the document (top-level) element. The XPath query

```
/site/open_auctions/open_auction[bidder][reserve]/seller
```

selects `seller` nodes that are children of `open_auction` nodes that have both a `bidder` and `reserve` child. It is easy to see that this query is satisfiable on documents valid with respect to the above DTD fragment.

In the full DTD, a `description` can occur as a descendant of more than one element, so one might write

```
/site//description[text][parlist]
```

to retrieve all `description` nodes that have both `text` and `parlist` children. However, this query is unsatisfiable with respect to the DTD, because a `description` can have only one of `text` or `parlist` as a child, not both.

Hidders investigates the satisfiability of XPath 2.0 expressions [8]. Various XPath fragments are classified as either in PTIME or NP-hard. Deciding satisfiability of XPath expressions in the context of a DTD/schema is one of the open problems mentioned in that paper. This problem is dealt with in [9], which considers a tree pattern formalism with expressiveness incomparable to XPath. The language expresses positive tree pattern queries, with data value equality and inequality along with a node-equality test. It shows that the satisfiability problem is NP-complete for several restrictions of this pattern language in the absence of DTDs. It also identifies cases in which satisfiability of tree pattern queries together with additional constraints, with and without schema, can be solved in polynomial time and develops algorithms for this purpose.

The most relevant work to ours is [2]. The authors consider a variety of XPath fragments widely used in practice, and investigate the impact of different XPath

[3] `http://monetdb.cwi.nl/xml/`

operators on satisfiability analysis. They 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. They show that with negation the complexity ranges from PSPACE to EXPTIME. When both data values and negation are in place, they find that the complexity ranges from NEXPTIME to undecidable. These results are extended in [6], where the satisfiability problem for a variety of XPath fragments with sibling axes is investigated, in the presence and the absence of DTDs and under various restricted DTDs. In these settings they establish complexity bounds ranging from NLOGSPACE to undecidable. They 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.

We concentrate on the satisfiability problem under two special classes of DTDs for a variety of XPath fragments with child axis (/), descendant axis (//), qualifiers ([ ]), wildcard (*) and union ($\cup$). The two classes of DTDs we consider are *duplicate-free* DTDs and *covering* DTDs. Informally, a duplicate-free DTD is one in which no regular expression uses the same symbol more than once. A covering DTD, on the other hand, is one in which each regular expression $R$ is such that the language $L(R)$ it denotes contains a string in which all the symbols used in $R$ appear. Formal definitions of these properties are provided in Section 2.

*Example 2.* All of the DTD rules for the XMark DTD fragment shown in Example 1 are covering, except the following

```
description   (text | parlist)
```

since the language denoted by `(text | parlist)` does not contain a sequence that includes both element names. In addition, all of the rules in the XMark fragment are duplicate-free. The following is an example of a rule with duplicates, taken from the XML Schema DTD[4] after replacing entity references and ignoring namespace prefixes

```
schema ((include | import | redefine | annotation)*,
        ((simpleType | complexType | element | attribute
          | attributeGroup | group | notation), (annotation)*)* )
```

where the element name `annotation` is repeated.

The notion of a duplicate-free DTD was introduced in [13] and also used in [14]. Other authors have also studied and classified DTDs according to various properties [3, 4]; however, the properties investigated are incomparable with the notions of covering and duplicate-free. We show in this paper that the classes of covering and duplicate-free DTDs comprise most real-world DTDs, i.e. those used in real-world applications. We identify a number of XPath fragments for which the complexity of the satisfiability problem reduces to PTIME

[4] `http://www.w3.org/2001/XMLSchema.dtd`

when duplicate-free or covering DTDs are used. For example, the fact that satisfiability for the fragment $\mathrm{XP}^{\{/,[\,]\}}$is NP-hard in general follows from a result in [13]. Here we show that it becomes decidable in PTIME for duplicate-free DTDs, although results from [2] imply that it remains NP-hard for the fragments $\mathrm{XP}^{\{/,[\,],*\}}$and $\mathrm{XP}^{\{[\,],//\}}$. More significantly, for covering DTDs we show that satisfiability for the fragment $\mathrm{XP}^{\{/,[\,],*,//,\cup\}}$is in PTIME.

The next section contains the definitions of the various forms of DTD and fragments of XPath we consider in this paper, and reviews the problem of XPath satisfiability in the presence of DTDs. Section 3 provides the results of our investigation into the relative frequency of covering and duplicate-free real-world DTDs. Section 4 presents our complexity results for a number of XPath fragments under duplicate-free and covering DTDs. Finally, Section 5 summarizes our main results and explains our future plans.

## 2 Notation and Background Material

In this section, we define DTDs, along with various subclasses of DTDs, the XPath fragments studied in this paper, and the notion of XPath satisfiability.

**Definition 1.** *A DTD over a finite alphabet $\Sigma$ is a tuple $(D, S_0, \Sigma)$ where $S_0 \in \Sigma$ is the start symbol, and $D$ is a mapping from $\Sigma$ to a set of regular expressions over $\Sigma$. We say that $R$ is the* content model *for symbol $a$ and write $a \rightarrow R$ (which we also call a* rule*). From now on, we refer to a DTD by $D$ rather than $(D, S_0, \Sigma)$ and assume that $\Sigma$ is the set of symbols appearing in $D$. In examples, we will usually drop the arrow symbol from rules.*

We will use the DTD syntax for regular expressions, namely, "," for concatenation, "|" for alternation (disjunction), "$*$" for reflexive transitive closure, "+" for transitive closure and "?" for optional.

**Definition 2.** *Let $R$ be a regular expression and $\Sigma$ be the set of symbols appearing in $R$. We say that $R$* covers *$\Sigma$, or simply that $R$ is* covering*, if there is a string in $L(R)$ that contains every symbol in $\Sigma$. A DTD $D$ is called* covering *if and only if each content model in $D$ is covering.*

Note that a number of common content models used in DTDs are covering. For example, the content models one gets from the naive representation of relational data as XML are covering, as are the so-called mixed content models found in "document-oriented" XML. Examples of covering and non-covering content models were given in Example 2.

**Definition 3.** *Let $R$ be a regular expression and $\Sigma$ be the set of symbols appearing in $R$. We say that $R$ is* duplicate-free *if each symbol in $\Sigma$ occurs exactly once in $R$. A DTD $D$ is called* duplicate-free *if and only if each content model in $D$ is duplicate-free.*

Note that the above definition is syntactic. In other words, we can have two regular expressions which denote the same language such that one expression is duplicate-free while the other is not. For example, $a?, b$ and $(a, b)|b$ denote the same language, but only the former expression is duplicate-free. Other examples of duplicate-free and non-duplicate-free expressions were given in Example 2.

A number of other subclasses of DTDs have been defined in order to study the complexity of problems such as XPath satisfiability. For example, [2] considers disjunction-free DTDs, while [3] considers simple regular expressions defined as follows.

**Definition 4.** *A* base symbol *is a regular expression $a$, $a?$ or $a^*$ where $a \in \Sigma$; a* factor *is of the form $e$, $e^*$ or $e?$ where $e$ is a disjunction of base symbols. A* simple regular expression *is $\epsilon$, $\emptyset$ or a sequence of factors.*

Clearly, a simple regular expression need not be duplicate-free nor covering. On the other hand, $a|(b, c)$ is duplicate-free but not simple, and $(a, b)^*$ is covering but not simple. We conclude that the 3 subclasses of DTDs are pairwise incomparable.

**Definition 5.** *The syntax of XPath expressions used in this paper is given by the following grammar:*

$$\begin{array}{l} q \rightarrow \text{`/'}\ p \\ p \rightarrow p\ \text{`/'}\ p \mid p\ \text{`//'}\ p \mid p\ \text{`}\cup\text{'}\ p \mid p\ \text{`['}\ p\ \text{`]'} \mid \text{`*'} \mid n \mid \text{`.'} \end{array}$$

*where $q$ is the start symbol, $n$ is an element name and '.' refers to the context node.*

Examples of XPath expressions using the above syntax were given in Example 1. As mentioned earlier, fragments of XPath are denoted by indicating which operators are supported. So the above fragment is denoted by $\text{XP}^{\{/,[\,],*,//,\cup\}}$, since child axis (/), descendant axis (//), qualifiers ([ ]), wildcard (*) and union ($\cup$) are all permitted.

Papers such as [2] use an alternative syntax where $\downarrow$ denotes use of the child axis without specifying an element name. So $\downarrow$ in their syntax corresponds to $*$ in ours, with $\downarrow /a$ corresponding to $a$. One consequence of this is that $*$ is implicitly permitted in all the XPath fragments considered by [2] that include the child axis, whereas we distinguish explicitly whether or not $*$ is included.

**Definition 6.** *The following notation is adapted from [2]. An XPath expression $p$ is* satisfiable *if there is an XML tree $T$ such that the answer of $p$ on $T$ is not empty, denoted by $T \models p$. Given a DTD $D$, we denote the fact that an XML tree* satisfies *(or is* valid *with respect to) $D$ by $T \models D$. Given a DTD $D$ and a query $p$, an XML tree $T$* satisfies *$p$ and $D$, denoted by $T \models (p, D)$, iff $T \models p$ and $T \models D$. For an XPath fragment $\mathcal{X}$, the* XPath satisfiability problem *$SAT(\mathcal{X})$ is, given a DTD $D$ and a query $p$ in $\mathcal{X}$, is there an XML tree $T$ such that $T \models (p, D)$.*

## 3 Real-World DTDs

In this section, we report on our investigation of “real-world” DTDs, i.e. those frequently used in real applications. For the purpose of this paper, we were concerned with two features of such DTDs, namely whether or not they were duplicate-free or covering. We will see that most of the real-world DTDs we studied have at least one of these two properties.

In order to examine the frequency of covering and duplicate-free DTD rules in real-world applications, we obtained 27 real-world DTDs, 13 using the Google search engine and 14 from the XML Data Repository[5]. The DTD names and a brief description of their application domains are given in Table 1.

| **DTD Name** | **Application Domain** |
|---|---|
| Oagis | Open Applications Group Archives |
| ODML 1.0 | Optimal Design Markup Language |
| LevelOne | HL7 Clinical Document Architecture |
| Ecoknowmics | Economic Knowledge Management |
| XML Schema | XML Schema |
| HP | HL7 Document Architecture |
| Meerkat | Storehouse of News about Technological Developments |
| OSD | Open Software Description |
| Opml | Outline Processing Markup Language |
| Rss-091 | XML Vocabulary for Describing Metadata about Websites |
| TV-Schedule | TV Schedule |
| Xbel-1.0 | XML Bookmarks Exchange Language |
| XHTML1-strict | Extensible HTML version 1.0 Strict |
| Newspaper | Newspaper |
| DBLP | Digital Bibliography Library Project |
| Music ML | Music Digital Library |
| XMark DTD | XML Benchmark |
| Yahoo | Yahoo Auction Data |
| Reed | Courses from Reed College |
| Nlm Medline | National Library of Medicine |
| SigmodRecord | Index of Articles from SIGMOD Record |
| Ubid | UBid Auction Data |
| Ebay | EBay Auction Data |
| News ML | News |
| PSD | Protein Sequence Database |
| Mondial 3.1 | World Geographic Database |
| 321gone | Auction |

**Table 1.** The DTDs and their application domains

We classified the content models into four separate groups as shown in Table 2. The first and the second columns of Table 2 show, respectively, the DTD

[5] `http://www.cs.washington.edu/research/xmldatasets`

| **DTD** | **Number** | **Covering** | | **Non-covering** | |
|---|---|---|---|---|---|
| **Name** | **of Rules** | Duplicate-free | Duplicates | Duplicate-free | Duplicates |
| Oagis | 617 | 422 | 161 | 16 | 18 |
| ODML 1.0 | 85 | 84 | 0 | 1 | 0 |
| LevelOne | 31 | 29 | 0 | 2 | 0 |
| Ecoknowmics | 224 | 221 | 1 | 2 | 0 |
| XML Schema | 26 | 19 | 1 | 6 | 0 |
| HP | 59 | 59 | 0 | 0 | 0 |
| Meerkat | 14 | 14 | 0 | 0 | 0 |
| OSD | 15 | 14 | 0 | 1 | 0 |
| Opml | 15 | 15 | 0 | 0 | 0 |
| Rss-091 | 24 | 24 | 0 | 0 | 0 |
| TV-Schedule | 10 | 10 | 0 | 0 | 0 |
| Xbel-1.0 | 9 | 9 | 0 | 0 | 0 |
| XHTML1-strict | 77 | 74 | 1 | 2 | 0 |
| Newspaper | 7 | 7 | 0 | 0 | 0 |
| DBLP | 37 | 37 | 0 | 0 | 0 |
| Music ML | 12 | 9 | 3 | 0 | 0 |
| XMark DTD | 77 | 76 | 0 | 1 | 0 |
| Yahoo | 32 | 32 | 0 | 0 | 0 |
| Reed | 16 | 16 | 0 | 0 | 0 |
| Nlm Medline | 41 | 41 | 0 | 0 | 0 |
| SigmodRecord | 11 | 11 | 0 | 0 | 0 |
| Ubid | 32 | 32 | 0 | 0 | 0 |
| Ebay | 32 | 32 | 0 | 0 | 0 |
| News ML | 116 | 112 | 0 | 4 | 0 |
| PSD | 66 | 64 | 0 | 2 | 0 |
| Mondial 3.1 | 23 | 23 | 0 | 0 | 0 |
| 321gone | 32 | 32 | 0 | 0 | 0 |
| **Total** | 1740 | 1518 | 167 | 37 | 18 |
| **Percentage** | 100% | 87.3% | 9.6% | 2.1% | 1.0% |

**Table 2.** The classification of DTD rules

names, and the number of rules in each DTD. The last four columns show, respectively, the number of rules that are (i) covering and duplicate-free, (ii) covering with duplicates, (iii) non-covering but duplicate-free, and (iv) non-covering with duplicates.

A quick glance at Table 2 reveals that the majority of the rules (87.3%) in these applications possess both the covering and duplicate-free properties. Most of the rest (11.7%) have exactly one of these properties, and only 1.0% of the rules are neither covering nor duplicate-free.

It should be pointed out that the 3 rules from the Music ML DTD[6] that contain duplicates are as follows:

[6] `http://xml.coverpages.org/musicML-DTD.txt`

| | Duplicate-free | Duplicates |
|---|---|---|
| Covering | 47 | 8 |
| Non-covering | 28 | 17 |

**Table 3.** The number of DTDs (out of 100) in each of the four categories

```
musicrow     ((entrysegment, segment+) | (entrysegment, segment+, text))
entrysegment ((entrypart) | (entrypart, entrypart))
segment      ((subsegment) | (subsegment, subsegment))
```

These rules are not even "unambiguous" as required by the XML specification. They can, however, easily be rewritten to be unambiguous as follows

```
musicrow     (entrysegment, segment+, text?)
entrysegment (entrypart, entrypart?)
segment      (subsegment, subsegment?)
```

resulting in the first rule becoming duplicate-free as well.

In further experiments, we obtained 73 more DTDs using Google and classified the content models into the same four groups. There were 3794 rules in total in these 73 DTDs and the majority of the rules (93.2%) possessed both the covering and duplicate-free properties. Most of the rest (6.1%) had exactly one of these properties, and only 0.7% of the rules were neither covering nor duplicate-free.

Because of our assumption that even a single rule that is non-covering (or contains duplicates) results in the DTD being classified as non-covering (or not duplicate-free), we need to determine which DTDs, as opposed to which rules, are covering (duplicate-free). Table 3 classifies the examined 100 DTDs into the four categories of covering and duplicate-free (top left), covering with duplicates (top right), non-covering but duplicate-free (bottom left) and non-covering with duplicates (bottom right). As shown in Table 3, the largest number (47%) of DTDs possess both properties, with only 17% possessing neither property. These experiments suggest, as a rule of thumb, that most real-word DTDs should be covering or duplicate-free. In fact, about 55% of the DTDs examined in the experiments were covering, and about 62% of the remaining (non-covering) DTDs (i.e. 28% of the whole) were duplicate-free. That is, 83% of the examined DTDs possessed at least one of the properties of being covering or duplicate-free.

## 4 XPath Satisfiability under Real-World DTDs

In this section, we concentrate on the satisfiablity problem of XPath queries under "real-world" DTDs, i.e. those which are either duplicate-free or covering. We will see that although the satisfiability problem is NP-complete or worse for many XPath fragments under general DTDs, it is in PTIME for certain XPath fragments when the underlying DTDs have the duplicate-free or covering properties.

### 4.1 XPath Satisfiability under Duplicate-free DTDs

Recall that a DTD is duplicate-free if each element name appears at most once in each content model. The fact that duplicate-free DTDs are easier to analyze was previously noted in [14], where it is shown that deciding containment under a DTD even for $\mathrm{XP}^{\{/,[\,]\}}$is coNP-complete, but that it reduces to PTIME when the DTD is duplicate-free. Below we show that the analysis of XPath satisfiability under duplicate-free DTDs is also simpler for certain fragments.

Before doing so, we state the following straightforward result.

**Proposition 1.** *Given a DTD $D$, deciding whether $D$ is duplicate-free can be done in PTIME.*

Benedickt *et al.* show that, in general, $\mathrm{SAT}(\mathrm{XP}^{\{/,[\,],*\}})$ and $\mathrm{SAT}(\mathrm{XP}^{\{[\,],//\}})$ are both NP-hard [2]. In fact, a result in [13] implies that $\mathrm{SAT}(\mathrm{XP}^{\{/,[\,]\}})$ is also NP-hard. However, we have the following result for duplicate-free DTDs.

**Theorem 1.** *Under duplicate-free DTDs, SAT($XP^{\{/,[\,]\}}$) is in PTIME.*

*Proof.* To prove the theorem, we first present two lemmas:

**Lemma 1.** *Let $R$ be a duplicate-free regular expression and $C$ be a nonempty set of symbols appearing in $R$. Then there is a string $w_c$ in $L(R)$ which covers all the symbols in $C$ if and only if every subexpression $(R_1|R_2)$ of $R$, where both $R_1$ and $R_2$ contain a symbol in $C$, appears as a subexpression of $(R_3)^*$ or $(R_3)^+$ for some $R_3$.*

*Proof.* It is important to note firstly that $R$ is duplicate-free and that $R$ is assumed to use every symbol in $C$. Therefore, each symbol in $C$ appears exactly once in $R$.

Assume $R$ contains a subexpression $(R_1|R_2)$ to which no closure operator applies, such that $R_1$ contains $c_1 \in C$ and $R_2$ contains $c_2 \in C$. Then each string in $L(R)$ containing $c_1$ has to exclude $c_2$ and each string in $L(R)$ containing $c_2$ has to exclude $c_1$, which means that no string in $L(R)$ covers $C$.

Conversely, assume there is no string in $L(R)$ which covers $C$. Since all the symbols in $C$ appear in $R$, there must be a pair of distinct symbols $c_1$ and $c_2$ in $C$ such that there are strings $w_1$ and $w_2$ in $L(R)$ such that $c_1$ (but not $c_2$) appears in $w_1$ and $c_2$ (but not $c_1$) appears in $w_2$, but no string in $L(R)$ contains both $c_1$ and $c_2$. Hence there must be a subexpression $(R_1|R_2)$ in $R$ such that $c_1$ appears in $R_1$ (or $R_2$) and $c_2$ appears in $R_2$ (respectively $R_1$). Furthermore, the expression $(R_1|R_2)$ cannot be subject to a closure operator; otherwise there would be a string in $L(R)$ containing both $c_1$ and $c_2$. □

**Lemma 2.** *Let $p$ be a two level XPath query in the fragment $XP^{\{/,[\,]\}}$such that the root $v_{root}$ has $n \geq 0$ leaf children. Then the satisfiability of $p$ under a duplicate-free DTD $D$ can be decided in PTIME.*

*Proof.* Let $R_{root}$ be the regular expression representing the content model in the DTD $D$ for the root node $v_{root}$ of $p$. In the case that more than one child of $v_{root}$ has the same label, say $b$, either the symbol $b$ is subject to some closure operator in $R_{root}$ (i.e. "$*$" or "$+$" applies to $b$ or to a term in which $b$ is contained) or not. In the former case, $v_{root}$ can have a $b$-child (while $D$-consistent) if, and only if, it can have many of them, and in the latter case all such $b$-children must map to the same $b$-node in any document tree which satisfies $D$ anyway. Therefore, we only need to check whether $L(R_{root})$ contains a word $w_c$ which includes all of the labels in $C$ (with an arbitrary ordering), where $C$ is the set, as opposed to the multiset, of labels of the children of $v_{root}$.

For each symbol $b$ in $C$, if there is no symbol $b$ in $R_{root}$, which can be checked in PTIME, then the answer (to whether $L(R_{root})$ contains $w_c$) is false. Otherwise, Lemma 1 applies and we only need to check whether $R_{root}$ contains some expression $(R_1|R_2)$ to which no closure operator applies and such that both $R_1$ and $R_2$ contain some symbol in $C$. The number of "$|$" operators in $R_{root}$ is $\mathrm{O}(|R_{root}|)$ and to obtain each expression $(R_1|R_2)$ in $R_{root}$ requires $\mathrm{O}(|R_{root}|)$ time. For each expression $(R_1|R_2)$ in $R_{root}$, to check whether $R_i$, $i = 1, 2$, covers some symbol in $C$ requires $O(|R_i| \times |C|)$ time. □

We now prove the theorem:

Let $p$ be a given XPath query in the fragment $\mathrm{XP}^{\{/,[]\}}$. For each internal node $v$ in $p$, if $v$ has more than one child with the same label, say $b$, then there are two possibilites: (i) the symbol $b$ is subject to some closure operators in the regular expression, say $R_v$, corresponding to the content model of $v$ in the DTD (i.e. "$*$" or "$+$" applies to $b$ or to a term in which $b$ is contained), or (ii) the symbol $b$ is not subject to a closure operator, hence all such $b$-children must map to the same $b$-node in any document tree which satisfies the DTD $D$ (because $D$ is duplicate-free). The latter case was previously called a *functional constraint* in[14] and was shown to be detectable in PTIME. In case(i), there is no restriction on the number of $b$-children of $label(v)$ (where $label(v)$ denotes the label of $v$) in any document tree which satisfies $D$, hence such $b$-children must not be merged. In case (ii), we merge all such $b$-children of $v$, which is done in PTIME. So we assume in the rest of the proof that case (ii) holds; that is, they must not be merged. Let $SubP(v)$ denote the two-level subtree of $p$ rooted at $v$. This implies that $SubP(v)$ has all the children of $v$ as leaves.

Based on the above terminology and the definition of satisfiablity, $p$ is satisfiable if and only if the subtree $SubP(v)$ is satisfiable for each internal node $v$ in $p$. Using Lemma 2, to decide whether such a subtree is satisfiable is in PTIME. On the other hand, there are altogether $m$ subtrees, where $m$ is the number of internal nodes, that is $m = O(|p|)$. Therefore, to decide whether $p$ is satisfiable is in PTIME. □

Even if a DTD as a whole is not duplicate-free, the above positive results can be used. For example, given a DTD $D$ and a query $p$ in $\mathrm{XP}^{\{/,[]\}}$, if every internal node in the tree representing $p$ is labelled by an element name whose content model is duplicate-free, then the satisfiability of $p$ can be determined in PTIME. This gives us the following.

**Corollary 1.** *Given a query $p$ in $XP^{\{/,[\,]\}}$and a DTD $D$, SAT($XP^{\{/,[\,]\}}$) is in PTIME if each rule in $D$ in which a symbol from $p$ appears is duplicate-free.*

We now consider the satisfiability of some other fragments of XPath under duplicate-free DTDs. The fact that SAT(XP$^{\{/,//,*\}}$) is in PTIME under duplicate-free DTDs follows trivially from a result in [2] showing that this fragment including union is in PTIME in general. However, we have the following negative results.

**Theorem 2.** *Under duplicate-free DTDs, the following problems are NP-hard:*

1. *SAT($XP^{\{/,[\,],*\}}$)*
2. *SAT($XP^{\{[\,],//\}}$)*
3. *SAT($XP^{\{/,[\,],\cup\}}$)*

*Proof.* (1) Benedikt *et al.* show in [2] that SAT(XP$^{\{/,[\,],*\}}$) is NP-hard by reduction from the 3SAT problem. Given a well-formed Boolean formula $\phi = C_1 \wedge \cdots \wedge C_n$ over variables $x_1, \ldots, x_m$, they define a DTD $D$ (with start symbol $S$) and the following rules:

$$\begin{aligned}
&S \rightarrow X_1, \ldots, X_m\\
&X_i \rightarrow T_i | F_i, \text{for } i \in [1, m]\\
&T_j \rightarrow C_{j_1}, \ldots, C_{j_k} \text{ /* all clauses } C_{j_i} \text{ in which } x_j \text{ appears */}\\
&F_j \rightarrow C_{j_1}, \ldots, C_{j_k}\text{/* all clauses } C_{j_i} \text{ in which } \overline{x}_j \text{ appears */}
\end{aligned}$$

They then define a query $XP(\phi) = /S[*/*/C_1] \cdots [*/*/C_n]$ such that $\phi$ is satisfiable iff $(XP(\phi), D)$ is satisfiable. As the DTD rules are duplicate-free, we can deduce that SAT(XP$^{\{/,[\,],*\}}$) under duplicate-free DTDs is NP-hard.

(2) The proof is the same as the proof of (1), except that $XP(\phi)$ is defined as $XP(\phi) = /S[.//C_1] \cdots [.//C_n]$.

(3) Using the same approach as (1), Benedikt *et al.* show in [2] that SAT(XP$^{\{/,[\,],\cup\}}$) is NP-hard. The DTD rules used in the proof are the following:

$$\begin{aligned}
&S \rightarrow X\\
&X \rightarrow (X?), (T \mid F)
\end{aligned}$$

and $XP(\phi) = /S[XP(C_1)] \cdots [XP(C_n)]$, where $XP(C_i)$ is defined as follows:

- For each variable $x_i$ in $\phi$, $XP(x_i) = X^i/T$ and $XP(\bar{x_i}) = X^i/F$, where $X^i$ is the chain $X/\cdots/X$ of length $i$.
- For each clause $C_j$, $XP(C_j)$ is $C_j$ in which each $x_i$ is replaced by $XP(x_i)$ and each $\bar{x_i}$ is replaced by $XP(\bar{x_i})$.

As the DTD rules are duplicate-free, SAT(XP$^{\{/,[\,],\cup\}}$) under duplicate-free DTDs is NP-hard. □

### 4.2 XPath Satisfiability under Covering DTDs

Recall that the majority of DTDs studied in Section 3 were classified as covering. In this section we prove that SAT($\text{XP}^{\{/,[\,],*,//,\cup\}}$) is in PTIME under covering DTDs. We should first point out the following fact which follows directly from a result in [13].

**Proposition 2.** *Given a DTD D, deciding whether D is covering is NP-complete.*

However, since we expect query processors to have to deal with relatively few, known DTDs while answering large numbers of XPath queries, the cost of detecting the covering property will be a one-off cost for each DTD.

**Theorem 3.** *Under covering DTDs, SAT(*$XP^{\{/,[\,],*,//,\cup\}}$*) is in PTIME.*

*Proof.* We prove this by the same method as Benedikt *et al.* use in [2], where they show that SAT($\text{XP}^{\{/,[\,],*,//,\cup\}}$) under disjunction-free DTDs is in PTIME. Clearly, a disjunction-free DTD is a special case of a covering DTD. It turns out, however, that the same proof technique can be used, the only substantial difference being that $*$ is implicitly permitted in all the XPath fragments considered by [2] that include the child axis, whereas we distinguish explicitly whether or not $*$ is included. We also need to adapt the proof from [2] to account for the different syntax for XPath used in that paper.

Let $p$ be a query in ($\text{XP}^{\{/,[\,],*,//,\cup\}}$) and $D$ be a covering DTD. We first construct a DTD digraph $G(V, E)$, with the set $V$ of element names in $D$. $G$ is rooted at $S_0 \in V$, where $S_0$ is the start symbol of $D$. For simplicity and without loss of generality we assume that neither $D$ nor $p$ is empty. Let $a$ and $b$ be two distinct symbols in $V$. There is an edge in $E$ from vertex $a$ to vertex $b$ if and only if $b$ appears in the content model of $a$ in $D$. We start by compiling the list $L$ of all sub-queries of $p$, topologically ordered such that $p_1$ precedes $p_2$ in $L$ if $p_1$ is a sub-query of $p_2$. For each $p' \in L$ and element name $a$ in $D$ we use a variable $reach(p', a)$ to hold the set of all element names reachable from $a$ via $p'$ in the DTD graph $G$. These variables are initially set to $\emptyset$, except that $reach(S_0, S_0) = \{S_0\}$. We also use a variable $sat(p', a)$ to hold the truth value indicating whether or not $p'$ is satisfiable at $a$.

The decision algorithm is outlined as follows[7]:

1. For each $p' \in L$ (in the order of $L$) and element name $a$ in $D$, we compute $reach(p', a)$ and $sat(p', a)$, based on the structure of $p'$:
   (a) $p' = .$: then $reach(p', a) = \{a\}$;
   (b) $p' = l$: then $reach(p', a) = \{l\}$ if $l$ appears in the content model of $a$ in $D$;
   (c) $p' = *$: then $reach(p', a)$ is the set of element names which appear in the content model of $a$;
   (d) $p' = \epsilon$: then $reach(p', a)$ is the set of element names reachable from $a$ in $G$ ($\epsilon$ can only appear as the *empty* XPath step, as in $//$);

[7] The $\epsilon$ and $\downarrow^*$ used in [2] correspond to our '.' and $\epsilon$, respectively.

(e) $p' = p_1 \cup p_2$: then $reach(p', a) = reach(p_1, a) \cup reach(p_2, a)$;
(f) $p' = p_1/p_2$ : then $reach(p', a) = \bigcup_{b \in reach(p_1,a)} reach(p_2, b)$;
In all the cases above, $sat(p', a) = true$ iff $reach(p', a) \neq \emptyset$.
(g) $p' = .[p_1]$: $sat(p', a) = sat(p_1, a)$, and $reach(p', a) = \{a\}$ if $sat(p_1, a) = true$;

2. Return $sat(p, S_0)$, where $S_0$ is the root of $G$.

Since $p[p_1] = p/.[p_1]$, the inductive case for $p[p_1]$ is reduced to $p_1/p_2$ and $.[p_1]$.

The algorithm iterates over all sub-queries in $L$ and all element names in $D$. Hence, the main loop in the algorithm is executed at most $O(|p||D|)$ times. Each step in the loop takes at most $O(|D|)$ time. Hence, the worst-case time complexity is $O(|p||D|^2)$.

The proof that the algorithm returns true iff $(p, D)$ is satisfiable follows the method used in [2]. The same method works because of the fact that since $D$ is covering, $.[q_1][q_2] \cdots [q_n]$, e.g., is satisfiable at an element labelled $a$ if and only if each of $q_1, q_2, \ldots, q_n$ is satisfiable at an element labelled $a$. This is also true for duplicate-free DTDs [2], but is not true in general. □

**Corollary 2.** *Given a query p in XP$^{\{/,[],*,//,\cup\}}$and a DTD D, SAT(XP$^{\{/,[],*,//\}}$) is in PTIME if each rule in D in which a symbol from p appears is covering.*

## 5 Conclusion and Future Work

This paper was concerned with discovering properties of real-world DTDs and their impact on the satisfiability problem. The motivation behind this was the authors' belief that although common XPath problems are of high complexity, e.g. NP-hard, in general, real-world applications usually provide simpler structures under which such otherwise hard problems could be performed in PTIME.

In particular, we examined several real-world DTDs and discovered a new property, called covering, which most of them preserved. We observed that even the minority of the examined real DTDs which did not possess the covering property were duplicate-free. We showed that the satisfiability problem of the XPath fragment XP$^{\{/,[],*,//,\cup\}}$reduces to PTIME when the underlying DTD has the covering property. We also showed that the satisfiability of the fragment XP$^{\{/,[]\}}$reduces to PTIME when the underlying DTD is duplicate-free. These problems were previously shown to be NP-hard under general DTDs.

The presented work is just a starting point in the direction of discovering features of real-world applications and deriving low-cost algorithms for problems such as query satisfiability, containment, and minimization. Among possible avenues for further research in this regard are the following:

- The experimental results in this paper showed that most of the DTDs classified as non-covering (respectively having duplicates) were done so because of only a few rules being non-covering (respectively containing duplicates). This suggests that one might introduce the concept of *locally*, vs. *globally*, covering (respectively duplicate-free) DTDs. The satisfiability problem under locally-covering (respectively locally duplicate-free) DTDs may still be in PTIME if the given queries preserve certain features.

- One could investigate PTIME algorithms for XPath *containment* under covering and duplicate-free DTDs. Some results for containment of queries in $\mathrm{XP}^{\{/,[\,]\}}$under duplicate-free DTDs are given in [14].
- When examining the DTDs, we noticed that some of the DTD rules are classified as having duplicates because of patterns such as $(a, b)|(a, c, d)$ where $a$ is duplicated. However, such a pattern is equivalent to $a, (b|(c, d))$ which is duplicate-free. Considering such a semantic notion of duplicate-free would increase the percentage of real-world DTDs classified as duplicate-free.
- We believe that, by combining the methods for covering and duplicate-free DTDs, $\mathrm{SAT}(\mathrm{XP}^{\{/,[\,]\}})$ can be decided in PTIME if each rule in a DTD is either covering or duplicate-free. This would then be applicable to 92 out of the 100 DTDs covered by our experiments.

## References

1. S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Tree pattern query minimization. *The VLDB Journal*, 11:315–331, 2002.
2. M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. In *Proc. Twenty-fourth ACM Symp. on Principles of Databases Systems*, June 2005. To appear in *J. ACM.*
3. G. J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML schema: A practical study. In *Proc. Seventh Int. Workshop on the Web and Databases*, pages 79–84, 2004.
4. B. Choi. What are real DTDs like? In *Proc. Fifth Int. Workshop on the Web and Databases*, pages 43–48, 2002.
5. S. Flesca, F. Furfaro, and E. Masciari. On the minimization of XPath queries. In *Proc. 29th Int. Conf. on Very Large Data Bases*, pages 153–164, 2003.
6. F. Geerts and W. Fan. Satisfiability of XPath queries with sibling axes. In *Proc. 10th Int. Workshop on Database Programming Languages*, pages 122–137, 2005.
7. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. *ACM Trans. on Database Syst.*, 30(2):444–491, 2005.
8. J. Hidders. Satisfiability of XPath expressions. In *Proc. 9th Int. Workshop on Database Programming Languages*, September 2003.
9. L. Lakshmanan, G. Ramesh, H. Wang, and Z. Zhao. On testing satisfiability of tree pattern queries. In *Proc. 30th Int. Conf. on Very Large Data Bases*, pages 120–131, 2004.
10. G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, January 2004.
11. F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Logical Methods in Computer Science*, 2(3), 2006.
12. P. Ramanan. Efficient algorithms for minimizing tree pattern queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 299–309, 2002.
13. P. T. Wood. Minimising simple XPath expressions. In *Proc. Fourth Int. Workshop on the Web and Databases*, pages 13–18, 2001.
14. P. T. Wood. Containment for XPath fragments under DTD constraints. In *Proc. 9th Int. Conf. on Database Theory*, pages 300–314, 2003.