Satisfiability of XPath Expressions

Jan Hidders

University of Antwerp
Dept. of Mathematics and Computer Science
Middelheimlaan 1, BE-2020 Antwerp, Belgium
jan.hidders@ua.ac.be

Abstract

In this paper, we investigate the complexity of deciding the satisfiability of XPath 2.0 expressions, i.e., whether there is an XML document for which their result is nonempty. Several fragments that allow certain types of expressions are classified as either in PTIME or NP-hard to see which type of expression make this a hard problem. Finally, we establish a link between XPath expressions and partial tree descriptions which are studied in computational linguistics.

1 Introduction

XPath is a simple language for selecting a set of nodes in an XML tree and as such it is used in many other XML-related standards such as XSLT, XQuery, XML Schema, XLink and XPointer. The satisfiability problem for XPath expressions is relevant for all these applications because it allows the detection of expressions that are probably erroneous and query optimizations that remove expressions that always return an empty result.

The satisfiability problem is a special case of the containment problem which has already been studied quite extensively [1, 12, 13, 16, 17]. However, in these studies usually only fragments with forward axes are considered, in which case most expressions are trivially satisfiable. Therefore, we will consider fragments in which all axes are allowed, including those that depend upon document order. We give four small examples of conflicts that may then occur:

1. `self::a/self::b`
   This path looks for a node that has at the same time name `a` and `b`.

2. `child::a/child::*/parent::b`
   This path requires that the node in the result has name `a` and `b`.

3. `/child::*/parent::*/parent::*`
   This path looks for a parent of the root node, which cannot exist.

4. `/preceding::*`
   This path looks for a node that precedes the root, but such a node cannot exist since the root is always the first node in the document order.

The organization of this paper is as follows. The next section contains the definition of XPath expressions and their semantics. Section 3 introduces the notion of tree description graph which are a special case of partial tree descriptions as studied in computational linguistics. Section 4 discusses the relationship between these tree description graphs and XPath expressions. Section 5 introduces a specific string matching problem that will be used in the following sections to show NP-hardness. Section 6 presents some complexity lowerbounds for certain fragments of XPath and Section 7 presents some upperbounds. Finally, Section 8 summarizes and discusses the presented results.

2 Initial Definitions

We start with the definition of the data model which is a simplification and abstraction of the full XML data model [8] and restricts itself to the element nodes. For this and following definitions we postulate an infinite set of tag names Σ.

Definition 2.1 (XML Tree)**.** An *XML tree* is a tuple $T = (N, \lhd, r, \lambda, \prec)$ such that $(N, \lhd)$ is a finite directed graph where N represents the set of *element nodes* and $\lhd$ represents the *parent-child relationship* that defines a tree with root r, $\lambda : N \to \Sigma$ is a labeling of the nodes that gives the tag name of each node and $\prec$ is a strict total order[1] over N that represents the *document order* and defines a pre-order tree-walk, i.e.,

Proceedings of the 9th International Conference on Data Base Programming Languages (DBPL), Potsdam, Germany, September 6-8, 2003

[1] A strict total order is a binary relation that is irreflexive, transitive and total.

PTW1 every child is smaller than its parent, i.e., if $n_1 \lhd n_2$ then $n_1 \prec n_2$ for all $n_1, n_2 \in N$, and

PTW2 if two nodes are siblings then all descendants of the smaller sibling are smaller than the larger sibling, i.e., for all two nodes $n_1, n_2 \in N$ for which there is a node $n_3 \in N$ such that $n_3 \lhd n_1$ and $n_3 \lhd n_2$ it holds that if $n_1 \prec n_2$ and $n_1 \lhd^+ n_4$ then $n_4 \prec n_2$, where $\lhd^+$ denotes the transitive closure of $\lhd$.

In the following we let $\lhd^+$ denote the transitive closure of $\lhd$, and $\lhd^*$ the transitive and reflexive closure of $\lhd$. Next, we define the set of XPath expressions that we will consider. We will use a syntax in the style of [2] that abstracts from the official syntax [3] and is more suitable for formal presentations.

Definition 2.2 (XPath Expression)**.** The set of *XPath expressions* is defined by the following abstract grammar:

$$\begin{aligned} P \quad ::= \quad & \epsilon \mid \uparrow \mid \downarrow \mid \uparrow^* \mid \downarrow^* \mid \dot{\twoheadleftarrow} \mid \dot{\twoheadrightarrow} \mid \\ & \Uparrow \mid \Sigma \mid P/P \mid P[P] \mid \\ & P \cap P \mid P \cup P \mid P - P \end{aligned}$$

where ϵ represents the empty path or *self* axis, $\uparrow$ and $\downarrow$ represent the *parent* and *child* axis, $\uparrow^*$ and $\downarrow^*$ represent the *ancestor-or-self* and *descendant-or-self* axis, $\dot{\twoheadleftarrow}$ and $\dot{\twoheadrightarrow}$ represent the *preceding-sibling* and *following-sibling* axis, $\Uparrow$ represents the document root, p_1/p_2 represents the concatenation of p_1 and p_2, $p_1[p_2]$ represents a path p_1 with a *predicate* p_2 and finally $\cap$, $\cup$ and $-$ represent the set intersection, set union and set difference.

All remaining axes in XPath can be straightforwardly defined in terms of the given axes: (ancestor) $\uparrow^+ \equiv \uparrow/\uparrow^*$, (descendant) $\downarrow^+ \equiv \downarrow/\downarrow^*$, (preceding) $\twoheadleftarrow \equiv \uparrow^*/\dot{\twoheadleftarrow}/\downarrow^*$, (following) $\twoheadrightarrow \equiv \uparrow^*/\dot{\twoheadrightarrow}/\downarrow^*$. Also boolean expressions in predicates can be readily simulated: $p_1 \wedge p_2 \equiv (p_1/\Uparrow) \cap (p_2/\Uparrow)$, $p_1 \vee p_2 \equiv (p_1/\Uparrow) \cup (p_2/\Uparrow)$ and $\neg p_1 \equiv \Uparrow - (p_1/\Uparrow)$.

Based on [15] and [7] and similar to [2] we define the semantics of these expressions as follows:

Definition 2.3 (XPath Semantics)**.** Given an XML tree $T = (N, \lhd, r, \lambda, \prec)$ we define the *semantics of a path expression* p, $[\![p]\!]_T \subseteq N \times N$, such that $(n, n') \in [\![p]\!]_T$ iff one of the following applies: (1) if $p = \epsilon$ then $n = n'$, (2) if $p = \uparrow$ then $n' \lhd n$, (3) if $p = \downarrow$ then $n \lhd n'$, (4) if $p = \uparrow^*$ then $n' \lhd^* n$, (5) if $p = \downarrow^*$ then $n \lhd^* n'$, (6) if $p = \dot{\twoheadleftarrow}$ then $n' \prec n$ and there is an n'' such that $n'' \lhd n$ and $n'' \lhd n'$, (7) if $p = \dot{\twoheadrightarrow}$ then $n \prec n'$ and there is an n'' such that $n'' \lhd n$ and $n'' \lhd n'$, (8) if $p = \Uparrow$ then $n' = r$, (9) if $p = t \in \Sigma$ then $n = n'$ and $\lambda(n) = t$, (10) if $p = p_1/p_2$ then there is an n'' such that $(n, n'') \in [\![p_1]\!]_T$ and $(n'', n') \in [\![p_2]\!]_T$, (11) if $p = p_1[p_2]$ then $(n, n') \in [\![p_1]\!]_T$ and there is an n'' such that $(n', n'') \in [\![p_2]\!]_T$, (12) if $p = p_1 \cap p_2$ then $(n, n') \in [\![p_1]\!]_T$ and $(n, n') \in [\![p_2]\!]_T$, (13) if $p = p_1 \cup p_2$ then $(n, n') \in [\![p_1]\!]_T$ or $(n, n') \in [\![p_2]\!]_T$, and (14) if $p = p_1 - p_2$ then $(n, n') \in [\![p_1]\!]_T$ and $(n, n') \notin [\![p_2]\!]_T$.

Remark. The tag names steps of the form $t \in \Sigma$ behave as if they follow the *self* axis. This means that `a/b` corresponds to the conventional XPath expression `self::a/self::b` and *not* to the expression `child::a/child::b` as is the case for the so-called abbreviated XPath syntax. Consequently the XPath expression `child::a/ancestor::b` can be represented in our syntax as $\downarrow$`/a/`$\uparrow^+$`/b` .

Fragments of P are denoted as $\mathcal{P}_V$ where V is a subset of $\{\Uparrow, [\,], \cap, \cup, -\}$. In $\mathcal{P}$ only expressions that consist of the axes, Σ and P/P are allowed. With the subscripts $\Uparrow$, $[\,]$, $\cap$, $\cup$ and $-$ also expressions of the form $\Uparrow$, $P[P]$, $P \cap P$, $P \cup P$ and $P - P$ are allowed, respectively.

Finally, we define what it means for an XPath expression to be satisfiable.

Definition 2.4 (XPath Satisfiability)**.** An XPath expression p is called *satisfiable* if there is an XML tree T such that $[\![p]\!]_T$ is not empty.

Example 2.1. Given two distinct tag names a and b in Σ the following expressions are *not* satisfiable: a/b, $a[b]$, $a/\downarrow/\uparrow/b$, $\Uparrow/\dot{\twoheadleftarrow}$ and $a/\downarrow/\dot{\twoheadrightarrow}/\uparrow/b$.

3 Tree Description Graphs

Before we discuss the problem of deciding satisfiability of XPath expressions we consider the same problem for a related notion called *partial tree descriptions* which has been studied in computational linguistics [14, 5, 10, 4]. A partial tree description can be informally described as a formula in EFO (Existential First Order Logic) that quantifies over the nodes in the tree and uses the binary predicates $=$, $\lhd$, $\lhd^+$, $\lhd^*$ and $\prec$ and a special constant r (the root). Such formulas can be used to describe various properties of ordered trees and also to query such trees. For our purposes we will consider only formulas that have the conjunction as their only logical operation and extend them with unary predicates for each tag name $t \in \Sigma$. This leads to the notion of *tree description* graph.

Definition 3.1 (Tree Description Graph)**.** A *tree description graph* (TDG) is a tuple $D = (V, v_r, \Phi)$ with V a finite set of variables, v_r a special element in V that represents the root and Φ a set of atoms of the following forms: $t(v_1)$ with $t \in \Sigma$ denoting that v_1 is labelled with t, $v_1 = v_2$, $v_1 \lhd v_2$, $v_1 \lhd^* v_2$, $v_1 \lhd^+ v_2$, $v_1 \prec v_2$ with $v_1, v_2 \in V$.

Example 3.1. An example of a TDG is $D = (V, v_r, \Phi)$ with variables $V = \{v_r, v_1, \ldots, v_5\}$ and atoms $\Phi = \{v_r \lhd^* v_1, v_1 \lhd^+ v_2, v_1 \lhd^* v_3, v_1 = v_4, v_3 \lhd v_4, v_4 \prec v_5, v_2 = v_5, b(v_2), a(v_4)\}$. This tree description graph is

shown in Figure 1 where the = predicate is indicated with double lines and the $\prec$ predicate is indicated with a dotted line.

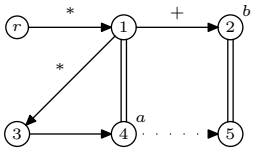

Figure 1: A tree description graph.

Tree description graphs can be seen as a generalization of *tree patterns* [12] that allows more axes. Like for XPath expressions we can also define a notion of satisfiability for TDGs.

Definition 3.2 (TDG Satisfiability)**.** Given a TDG $D = (V, v_r, \Phi)$ a *model for* D is a tuple $M = (T, I)$ with an XML tree $T = (N, \lhd, r, \lambda, \prec)$ and an interpretation $I : V \to N$ such that $I(v_r) = r$ and I makes all atoms in Φ satisfied for T. A TDG is called *satisfiable* if there is a model for it.

The TDG in Figure 1 is not satisfiable because in a model (T, I) it would both have to hold that $I(v_1) = I(v_4)$ and $I(v_1) \lhd^+ I(v_4)$, which is not possible.

Similar to [12] we show that when reasoning about a certain property of patterns we only need to consider a limited set of models; for deciding satisfiability we only need to consider models that have the same size as the TDG.

Lemma 3.1. *If a tree description graph* $D = (V, v_r, \Phi)$ *is satisfiable then there is a model* (T, I) *for* D *with at most* $|V|$ *nodes in* T*.*

Proof. We show that if for the TDG $D = (V, v_r, \Phi)$ there is a model $M_1 = (T_1, I_1)$ with XML tree $T_1 = (N_1, \lhd_1, r_1, \lambda_1, \prec_1)$ and $|N_1| > |V|$ then there is a model for D with one node less than M_1. By induction upon $|N_1|$ it then follows that the lemma holds.

The smaller model is $M_2 = (T_2, I_1)$ where T_2 is constructed as follows. We choose an arbitrary node n in N_1 that is not in the image of I_1, i.e., there is no $v \in V$ such that $I_1(v) = n$. Such a node exists since $|N_1| > |V|$. We then construct T_2 by (1) removing n from the tree and (2) making the children of n now the children of the parent of n. Note that n will always have a parent since the only node that has no parent is the root and the root is always in the image of I_1. More formally, we define $T_2 = (N_2, \lhd_2, r_2, \lambda_2, \prec_2)$ such that (1) $N_2 = N_1 - \{n\}$, (2) $\lhd_2 = (\lhd_1 \cap (N_2 \times N_2)) \cup \{(n', n'') | n' \lhd_1 n \wedge n \lhd_1 n''\}$, (3) $r_2 = r_1$, (4) $\lambda_2 = \lambda_1|_{N_2}$ where $\lambda_1|_{N_2}$ is the restriction of the function λ_1 to the domain N_2, and (5) $\prec_2 = \prec_1 \cap (N_2 \times N_2)$. Then it can be shown that T_2 is indeed an XML tree and (T_2, I_1) is a model for D:

T_2 is an XML Tree : By its construction T_2 defines a finite labelled tree. What remains to be shown is that $\prec$ defines a pre-order tree-walk. The condition PTW1 holds for T_2 because if $n_1 \lhd_2 n_2$ then by the construction of $\lhd_2$ it follows that $n_1 \lhd_1^+ n_2$ and so $n_1 \prec_1 n_2$ which implies $n_1 \prec_2 n_2$. The condition PTW2 also holds for T_2, which can be shown as follows. Assume that $n_3 \lhd_2 n_1$, $n_3 \lhd_2 n_2$, $n_1 \prec_2 n_2$ and $n_1 \lhd_2^+ n_4$. By the construction of T_2 it will hold that $n_1 \prec_1 n_2$ and $n_1 \lhd_1^+ n_4$. It will also hold for the removed node n that either (a) $n_3 \lhd_1 n \lhd_1 n_1$, or (b) $n_3 \lhd_1 n \lhd_1 n_2$, or (c) $n_3 \lhd_1 n_1$ and $n_3 \lhd_1 n_2$. In all cases we can derive that $n_4 \prec_1 n_2$ as follows. In case (a) it holds that in T_1 the node n was a smaller sibling of n_2 and therefore $n_4 \prec_1 n_2$. In case (b) it holds that in T_1 the node n was a larger sibling of n_1 and therefore $n_4 \prec_1 n$. By TW1 it also holds that $n \prec_1 n_2$, so it follows that $n_4 \prec_1 n_2$. In case (c) it directly follows by TW2 for T_1 that $n_4 \prec_1 n_2$. Summarizing we now know that in all cases $n_4 \prec_1 n_2$ and since both nodes are in N_2 it follows that $n_4 \prec_2 n_2$.

(T_2, I_1) is a model for D: Because $r_1 = r_2$ and λ_2 and $\prec_2$ are restrictions of λ_1 and $\prec_1$, respectively, to N_2 it follows that $I_1(v_r) = r_2$ and I_1 makes the t atoms and the $\prec$ atoms satisfied for T_2. Because $\lhd_2^+$ and $\lhd_2^*$ are equal to restrictions of $\lhd_1^+$ and $\lhd_1^*$ to N_2 it also holds for these atoms that I_1 makes them satisfied for T_2. Finally, because $\lhd_2$ is a superset of the restriction of $\lhd_1$ to N_2 it also holds that I_1 satisfies the $\lhd$ atoms for T_2. □

The previous observation then leads to the following theorem.

Theorem 3.2. *Deciding satisfiability of a tree description graph is in NP.*

Proof. It is easy to see that there is a model for a TDG $D = (V, v_r, \Phi)$ iff there is a model for $D' = (V', v_r, \Phi)$ where V' is the subset of V that is mentioned in Φ plus v_r. By Lemma 3.1 it holds that D' is satisfiable iff there is a model for D' with at most $|V'|$ nodes. If the size of the representation of D' is n then there can be no more than $2n$ variables in V'. It follows that D is satisfiable iff there is a model for D with a representation size of $\mathcal{O}(n^2)$. It follows that a nondeterministic algorithm can guess an XML tree T and an interpretation I in a polynomial number of steps. Since it can also be checked in polynomial time that (T, I) is a model of D' it follows that there is an NP algorithm that decides whether there is a model for D. □

4 TDGs and XPath

Both TDGs and XPath expressions can be used to define binary relations over the nodes of a given XML tree. For example, the binary relation defined by the path expression $(\downarrow^*/a/\uparrow/b/\uparrow^*) \cap (\downarrow^+/c/\dot{\twoheadrightarrow})$ is also defined by the TDG in Figure 2 where the begin and end

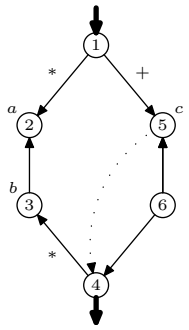

Figure 2: A TDG representing an XPath expression.

variable are indicated by a bold incoming and leaving arrow, respectively.

In general the translation of an XPath expression in the fragment $\mathcal{P}_{\Uparrow,[\,],\cap}$ to a TDG is defined as follows.

Definition 4.1 (Atom Set)**.** Given an XPath expression $p \in \mathcal{P}_{\Uparrow,[\,],\cap}$, a begin variable v and an end variable v' the *atom set for p from v to v'*, denoted as $\Phi_{v,v'}(p)$, is defined as follows:

$$
\begin{aligned}
\Phi_{v,v'}(\epsilon) &= \{v = v'\} \\
\Phi_{v,v'}(\uparrow) &= \{v' \lhd v\} \\
\Phi_{v,v'}(\downarrow) &= \{v \lhd v'\} \\
\Phi_{v,v'}(\uparrow^*) &= \{v' \lhd^* v\} \\
\Phi_{v,v'}(\downarrow^*) &= \{v \lhd^* v'\} \\
\Phi_{v,v'}(\dot{\twoheadleftarrow}) &= \{v' \prec v, v'' \lhd v, v'' \lhd v'\} \\
&\quad \text{with } v'' \text{ a fresh variable} \\
\Phi_{v,v'}(\dot{\twoheadrightarrow}) &= \{v \prec v', v'' \lhd v, v'' \lhd v'\} \\
&\quad \text{with } v'' \text{ a fresh variable} \\
\Phi_{v,v'}(\Uparrow) &= \{v' = v_r\} \\
\Phi_{v,v'}(t) &= \{v = v', t(v')\} \\
\Phi_{v,v'}(p_1/p_2) &= \Phi_{v,v''}(p_1) \cup \Phi_{v'',v'}(p_2) \\
&\quad \text{with } v'' \text{ a fresh variable} \\
\Phi_{v,v'}(p_1[p_2]) &= \Phi_{v,v'}(p_1) \cup \Phi_{v',v''}(p_2) \\
&\quad \text{with } v'' \text{ a fresh variable} \\
\Phi_{v,v'}(p_1 \cap p_2) &= \Phi_{v,v'}(p_1) \cup \Phi_{v,v'}(p_2)
\end{aligned}
$$

The correctness of this translation is established by the following theorem.

Theorem 4.1. *Given a path expression $p \in \mathcal{P}_{\Uparrow,[\,],\cap}$ and if V is the set of all variables in $\Phi_{v,v'}(p)$ plus v_r then for every XML tree T and nodes n, n' in T it holds that there is a model (T, I) for $(V, v_r, \Phi_{v,v'}(p))$ with $I(v) = n$ and $I(v') = n'$ iff $(n, n') \in [\![p]\!]_T$.*

Proof. (Sketch) This can be shown with induction upon the structure of p and follows straightforwardly from the given semantics of XPath expressions. □

Whether each TDG can be translated to an equivalent expression in the fragment $\mathcal{P}_{\Uparrow,[\,],\cap}$ is still an open problem.

It follows from this translation that deciding satisfiability of path expressions in $\mathcal{P}_{\Uparrow,[\,],\cap}$ is in NP.

Theorem 4.2. *Deciding satisfiability of path expressions in $\mathcal{P}_{\Uparrow,[\,],\cap}$ is in NP.*

Proof. Satisfiability of the path expression p can be decided by translating it to the corresponding TDG and deciding if this is satisfiable. The translation can be done in PTIME and by Lemma 3.2 we can decide satisfiability of TDGs in NP. □

5 String Matching Problems

In order to show the hardness of deciding satisfiability for certain XPath fragments we will show that the following string matching problem can be reduced to these problems.

Definition 5.1 (Bounded Multiple String Matching Problem)**.** Given a finite set of patterns A, which are strings over $\{0, 1, *\}$, is there a string over $\{0, 1\}$ whose size is equal to the size of the largest pattern in A and in which all patterns in A can be matched with $*$ as a wildcard for one symbol?

In the following we will also refer to this problem as the BMS problem.

Theorem 5.1. *Deciding the BMS problem is NP-complete.*

Proof. It is easy to see that such a string can be guessed and verified in non-deterministic polynomial time

To prove NP-hardness we show that there is a polynomial reduction from 3SAT [9] to this problem. The reduction consists of a mapping of a CNF formula to a set of formulas such that there is a string into which all these patterns can be matched and that is as large as the largest pattern iff this string encodes a truth assignment for the formula.

To demonstrate the principle we will first show how the formula $\varphi = C_1 \wedge C_2$ with $C_1 = X_1 \vee \neg X_2 \vee X_3$ and $C_2 = \neg X_1 \vee X_2 \vee \neg X_4$ is translated. The encoding of the truth assignment is illustrated by the first pattern in Figure 3, which is called a_{pre}.

The pattern a_{pre} will be the longest pattern and therefore defines the length of the string, in this case 62 characters. It also enforces that the string begins with `101010`. The underlined positions marked by $X_j^{(i)}$ are intended to encode a truth assignment for variable X_j in clause C_i; the pair `10` denotes *true* and `01` denotes *false*.

Because in the chosen encoding every clause has a separate truth assignment we introduce the patterns $a^1_{C_1,X_1}, a^0_{C_1,X_1}, \ldots, a^1_{C_1,X_4}, a^0_{C_1,X_4}$. These patterns are one character shorter than a_{pre} and each pattern $a^1_{C_1,X_j}$ ($a^0_{C_1,X_j}$) contains two 1s (0s); one at the first position of the pair marked with $X_j^{(1)}$ and the

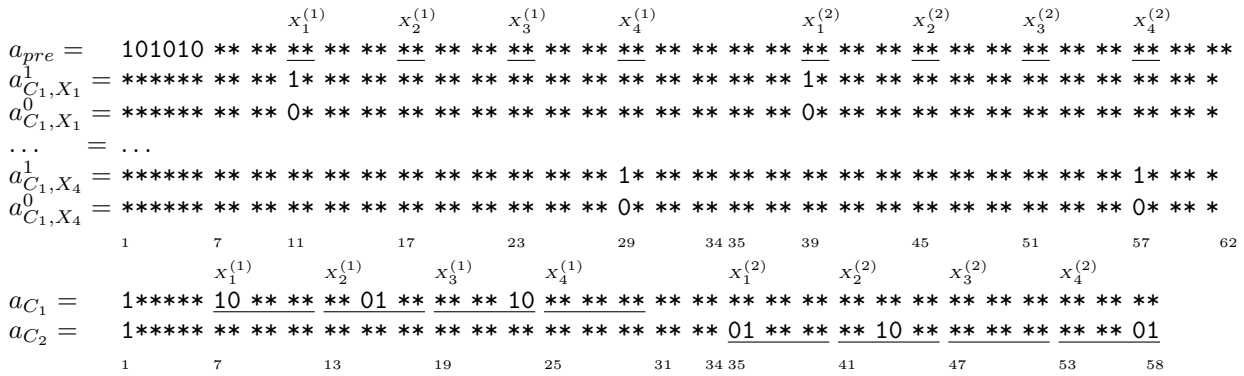

Figure 3: A set of patterns for the formula $C_1 \wedge C_2$ with $C_1 = X_1 \vee \neg X_2 \vee X_3$ and $C_2 = \neg X_1 \vee X_2 \vee \neg X_4$

other at $X^{(2)}_j$. Because of their length these patterns can only be embedded in the string in two ways; starting from the first or from the second position in the string. It is then easy to see that the patterns $a^1_{C_1,X_1}$ and $a^0_{C_1,X_1}$ enforce that the positions marked by $X^{(1)}_1$ and $X^{(2)}_1$ contain either both 10 or both 01. If we introduce such a pattern for each variable then we can ensure that all truth assignment for all clauses are the same.

Finally, we introduce the patterns a_{C_1} and a_{C_2} to ensure that the clauses C_1 and C_2, respectively, are satisfied. First note that these patterns start with 1 and their length is 4 less than the string. Since a_{pre} enforces that the string starts with 101010 it follows that these patterns can only be matched in three ways; starting from the first, third or fifth position. The construction of a_{C_1} is now as follows. For each variable X_j there is a region of six characters underlined and marked with $X^{(1)}_j$. In this region we set the kth pair to 10 if the variable appears in the kth position in the clause, and to 01 if the negation of the variable appears at that position. So for the clause $C_1 = X_1 \vee \neg X_2 \vee X_3$ the first pair of X_1's region is set to 10, the second pair in X_2's region is set to 01 and the third pair in X_3's regions is set to 10.

Now consider what happens with each way that this pattern can be matched in the string.

1. If it is matched from the *first* position in the string then the pair 10 at positions [23,24] is mapped to the same positions in the string which are marked as $X^{(1)}_3$ in a_{pre}, and all other 10 and 01 pairs are mapped to unmarked positions

2. If the pattern a_{C_1} is matched from the *third* position then the pair 01 at positions [15,16] is mapped to the positions [17,18] in the string which are marked as $X^{(1)}_2$ in a_{pre}, and all other 10 and 01 pairs are mapped to unmarked positions.

3. Finally, if the pattern a_{C_1} is matched from the *fifth* position then the pair 10 at positions [7,8] is mapped to the positions [11,12] in the string which are marked as $X^{(1)}_1$ in a_{pre}, and all other 10 and 01 pairs are mapped to unmarked positions.

Summarizing, if the pattern matches then the encoded truth assignment for $X^{(1)}_1, \ldots, X^{(1)}_4$ will make at least one literal in the clause C_1 true. In a similar fashion the pattern a_{C_2} ensures that the encoded truth assignment for $X^{(2)}_1, \ldots, X^{(2)}_4$ will make at least one literal in C_2 true.

We can now summarize the meaning of the patterns in Figure 3 as follows. The pattern a_{pre} defines a preamble and the lengt of the string such that there is room for a separate truth assignment for each clause. The patterns $a^1_{C_1,X_1}, a^0_{C_1,X_1}, \ldots, a^1_{C_1,X_4}, a^0_{C_1,X_4}$ ensure that all the truth assignments for the different clauses are in fact the same truth assignment. Finally, the patterns a_{C_1} and a_{C_2} ensure that the truth assignment for C_1 makes C_1 satisfied and the truth assignment for C_2 makes C_2 satisfied. It then follows that a string of the same length as a_{pre} into which all these patterns match, defines a truth assignment that makes φ satisfied.

On the other hand, if there is a truth assignment that makes φ satisfied then we can construct a string of the same length as a_{pre} such that all patterns match into this string as follows. As required by a_{pre} we let the string start with 101010 and we encode the truth assignment in the string in the positions that are marked for a_{pre}. Since we assign the same truth assignment for all clauses the patterns $a^1_{C_1,X_1}, a^0_{C_1,X_1}, \ldots, a^1_{C_1,X_4}, a^0_{C_1,X_4}$ will all match. Finally we can make sure that a_{C_1} and a_{C_2} match into the string by choosing for each clause one literal that is made true by the assignment and mapping it to the position in a_{pre} that is marked for that literal. Since all other 10 and 01 in the pattern will then be mapped to unmarked positions it follows that we can dan fill these positions in the string such that the pattern indeed matches. For example, if the truth assignment maps X_2 to *false* then we might map a_{C_1} into the string such that the positions [15,16] are mapped to

[17,18] in the string, and therefore positions [7,8] and [23,24] are mapped to [9,10] and [25,26], respectively, and therefore these positions in the string should contain the pairs at [7,8] and [23,24] in a_{C_1}, i.e., in both cases 10.

Summarizing, we have shown that the BSM problem defined by the patterns in Figure 3 is satisfiable iff the formula φ is satisfiable. This concludes the example and we will now proceed with a description of the reduction in general.

Let us consider a formula $\varphi = C_1 \wedge \ldots \wedge C_m$ with $C_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$ with $l_{i,k} = X_j$ or $l_{i,k} = \neg X_j$ where $X_1, \ldots, X_n$ are the variables in φ.

The first pattern defines the preamble and the length of the string:

$$a_{pre} \quad = \quad \texttt{101010}\ *^{(4+6n)m}$$

Note that the total length of this pattern is $6 + (4 + 6n)m$ and that the assignment of variable X_j for clause C_i can be found in the pair that starts at position $7 + (4 + 6n)(i-1) + 4 + 6(j-1)$ in the string.

We then proceed with constructing the patterns that ensure that the truth assignment for variable X_j in C_i is equal to that in C_{i+1}:

$$\begin{aligned} a^1_{C_i,X_j} \quad &= \quad ******\\ &\quad\ *^{(4+6n)(i-1)}\\ &\quad\ *^{4+6(j-1)}\texttt{1}*^{(3+6n)}\texttt{1}*^{(6n-2)-6(j-1)}\\ &\quad\ *^{(4+6n)(m-i-1)}\\ a^0_{C_i,X_j} \quad &= \quad ******\\ &\quad\ *^{(4+6n)(i-1)}\\ &\quad\ *^{4+6(j-1)}\texttt{0}*^{(3+6n)}\texttt{0}*^{(6n-2)-6(j-1)}\\ &\quad\ *^{(4+6n)(m-i-1)} \end{aligned}$$

Because the length of these two patterns is $6 + (4 + 6n)m - 1$ there are only two ways in which it can be matched with a string of length $6 + (4 + 6n)m$.

Finally we define the patterns that ensure that the encoded truth assignment make a certain clause satisfied. For this purpose we define $a[C_i, X_j]$ as equal to ****** except that the kth pair is equal 10 if $l_{i,k} = X_j$ and equal to 01 if $l_{i,k} = \neg X_j$. We then construct the patterns that ensure that the clause C_i is made satisfied by the truth assignment for C_i as follows.

$$\begin{aligned} a_{C_i} \quad &= \quad \texttt{1}*****\\ &\quad\ *^{(4+6n)(i-1)}\\ &\quad\ a[C_i, X_1] \ldots a[C_i, X_n]\\ &\quad\ *^{(4+6n)(m-i)} \end{aligned}$$

Note that a_{C_i} is padded with *s to a length of $2 + (4 + 6n)m$ to ensure that it can only be matched to the 1st, 3rd and 5th position in the string that has the length of a_{pre} and into which a_{pre} matches.

The total set of patterns for φ is now defined as

$$\begin{aligned} A_\varphi \quad &= \quad \{a_{pre}\} \cup\\ &\quad\ \{a^0_{C_i,X_j}, a^1_{C_i,X_j} \mid 1 \leq i < m, 1 \leq j \leq n\} \cup\\ &\quad\ \{a_{C_i} \mid 1 \leq i \leq m\} \end{aligned}$$

It can be shown that all the patterns in A_φ are bounded polynomially in n and m:

$$\begin{aligned} |a_{pre}| \quad &= \quad 6 + (4 + 6n)m\\ |a^b_{C_i,X_j}| \quad &= \quad 6 + (4 + 6n)m - 1\\ |a_{C_i}| \quad &= \quad 2 + (4 + 6n)m \end{aligned}$$

It follows that they are polynomially bounded by the size of φ and because there are $1 + 2(m-1)n + m$ patterns in A_φ it also holds that the representation of this set is bounded polynomially in the size of φ. Consequently it is easy to see that A_φ can be generated from φ in polynomial time.

What remains to be shown is that A_φ is satisfiable iff φ is satisfiable.

It is easy to see that if a string x satisfies A_φ then we can read a truth assignment that satisfies from the position for the truth assignment for X_j in any C_i. Because of the $a^b_{C_i,X_j}$ patterns these assignment will be the same for any clause C_i and because of the a_{C_i} patterns all clauses in φ will be satisfied.

If there is a truth assignment that satisfies φ then we can construct x as follows. We start the string with 101010 and fill in x the positions for X_j for each clause C_i as prescribed by x. This ensures that the a_{pre} pattern and the $a^b_{C_i,X_j}$ patterns are satisfied. Next, we pick in each clause one of the three literals that is satisfied by the truth assignment and map the a_{C_i} patterns accordingly to x and set the 1s and 0s that are required by them. Finally, the remaining positions in x can be filled with arbitrary 1s and 0s. □

6 Lower-Bound Results

We now proceed with discussing the hardness of deciding satisfiability of fragments of XPath. The first hardness result concerns $\mathcal{P}_\cap$, i.e., the fragment that allows only expressions that consist of the axes and expressions of the form Σ, P/P and $P \cap P$.

Theorem 6.1. *Deciding satisfiability of path expressions in $\mathcal{P}_\cap$ is NP-hard.*

Proof. We show that the BMS problem can be reduced to this problem. We assume that the set of patterns is $\{a_0, \ldots, a_n\}$ and that a_0 is the longest pattern. The pattern a_0 is translated to a path p_0 by translating a 0 to $\downarrow/a$, 1 to $\downarrow/b$ and $*$ to just $\downarrow$. For example, "*0*0*1" is translated to $\downarrow/\downarrow/a/\downarrow/\downarrow/a/\downarrow/\downarrow/b$. The other patterns a_i are translated to p_i in the same way but with an extra $\downarrow^*$ step before and after it. So, for example,

"10" is translated to $\downarrow^*/\downarrow/b/\downarrow/a/\downarrow^*$. Finally we take the intersection of all these paths: $p_0 \cap p_1 \cap \ldots \cap p_n$.

It is easy to see that if there is an XML tree T and a pair (n, n') in the semantics of this path under T then labels of the nodes in the path from n to n' represent a string into which all patterns match if we replace a and b with 1 and 0, respectively.

Conversely, if there is a string into which all pattern can be matched then we can construct an XML tree that consists of a simple path that is labelled with the labels that correspond with the characters in the string, for which the semantics of the path expression will contain at least (n, n') with n and n' the begin and end node of this path, respectively. □

Remark. In the proof we only need the forward axes $\downarrow$ and $\downarrow^*$ and the ordering of the trees is not used.

Theorem 6.2. *Deciding satisfiability of tree description graphs is NP-hard.*

Proof. This follows from the straightforward translation of path expressions in $\mathcal{P}_{\Uparrow,[\,],\cap}$ as given in Definition 4.1 and Theorem 6.1. □

Theorem 6.3. *Deciding satisfiability of path expressions in $\mathcal{P}_-$ is NP-hard.*

Proof. This proof proceeds similar to the one of Theorem 6.1 except that the path $p_0 \cap p_1 \cap \ldots \cap p_n$ is simulated with $p_0 - (p_0 - p_1) - (p_0 - p_2) - \ldots - (p_0 - p_n)$. □

Theorem 6.4. *Deciding satisfiability of path expressions in $\mathcal{P}_{[\,],\cup}$ is NP-hard.*

Proof. We show this by reducing the problem SAT [9]. We construct for every CNF formula $\varphi = C_1 \wedge \ldots \wedge C_m$ a path p_φ as follows. Let the variables in φ be $X_1, \ldots, X_n$. For every literal l we define a path p_l such that p_{X_i} is a path of $n+1$ steps of the form $\uparrow$ except step $i+1$ which is of the form a, and $p_{\neg X_i}$ is the same except that step $i+1$ is of the form b. For example, for $n = 3$:

$$\begin{aligned} p_{X_2} &= \uparrow/\uparrow/a/\uparrow \\ p_{\neg X_3} &= \uparrow/\uparrow/\uparrow/b \end{aligned}$$

A clause $l_1 \vee \ldots \vee l_p$ is straightforwardly mapped to $p_{l_1} \cup \ldots \cup p_{l_p}$. For example $p_{X_2 \vee \neg X_3}$ is

$$(\uparrow/\uparrow/a/\uparrow) \cup (\uparrow/\uparrow/\uparrow/b)$$

Finally, the formula $C_1 \wedge \ldots \wedge C_m$ is mapped to $\epsilon[p_{C_1}]\ldots[p_{C_m}]$.

It is easy to see that p_φ is satisfiable iff φ is satisfiable. Moreover, if k is the length of φ then $m \leq k$, $n \leq k$ and k will also be the upper-bound for the number of literals per clause, and therefore the size of p_φ will be in $\mathcal{O}(k^3)$. □

Theorem 6.5. *Deciding satisfiability for $\mathcal{P}_{\Uparrow,[\,]}$ is NP-hard.*

Proof. Similar to the proof of Theorem 6.1 we show that the BMS problem can be reduced to this problem. We assume that the set of patterns is $\{a_0, \ldots, a_n\}$ and that a_0 is the longest pattern. The pattern a_0 is translated to a path p_0 by starting with a $\Uparrow$ followed by translating a 0 to $\downarrow/a$, 1 to $\downarrow/b$ and $*$ to just $\downarrow$. For example, "*0*0*1" is translated to $\Uparrow/\downarrow/\downarrow/a/\downarrow/\downarrow/a/\downarrow/\downarrow/b$. The other patterns a_i are translated to p_i in a similar fashion but here we start with $\uparrow^*$ and then translate the pattern *in reverse* and with the $\uparrow$ axis in stead of the $\downarrow$ axis. For example, "11*0" is translated to $\uparrow^*/\uparrow/b/\uparrow/\uparrow/a/\uparrow/a$. Finally we construct from these paths the following path: $p_0[p_1][p_2]\ldots[p_n]$.

As in the proof in Theorem 6.1 it holds for this path expression that if there is a pair in its semantics for a certain XML tree then the labels of the nodes in the path between those nodes corresponds to a string that satisfies the original BSM problem. Conversely, if there is a string that satisfies the BSM problem then a path that is labelled correspondingly and starts from the root will constitute an XML tree that satisfies the path expression. □

Remark. Unlike the proof of Theorem 6.1 this one requires forward axes ($\downarrow$) and backward axes ($\uparrow^*$ and $\uparrow$), but still does not use axes based on document order.

7 Upper-Bound Results

In this section we discuss some upper-bounds for XPath fragments. For the very large fragment $\mathcal{P}_{\Uparrow,[\,],\cap,\cup}$ that allows everything except the set difference, it can be shown that deciding satisfiability is in NP.

Theorem 7.1. *Deciding satisfiability of path expressions in $\mathcal{P}_{\Uparrow,[\,],\cap,\cup}$ is in NP.*

Proof. The algorithm starts with guessing nondeterministically for every subexpression of the form $p_1 \cup p_2$ if it replaces it with just p_1 or p_2 and for the resulting $\mathcal{P}_{\Uparrow,[\,],\cap}$ expression it decides with the algorithm of Theorem 4.2 if the resulting $\mathcal{P}_\cap$ expression is satisfiable. □

Remark. At the moment we don't have an upper bound for $\mathcal{P}_-$ and it is not even known if it decidable.

Theorem 7.2. *Deciding satisfiability for $\mathcal{P}_{[\,]}$ is in PTIME.*

Proof. (Sketch) We start with using Definition 4.1 to transform the path expression to a TDG. The result will be essentially a tree except for small cycles of three nodes to simulate the $\dot{\rightarrow}$ and $\dot{\leftarrow}$ axes.

We then apply the following rules to this graph until they can be applied no more:

1. If there is an atom $v_i = v_j$ then it is removed and all occurrences of v_i are replaced by v_j, i.e., the nodes v_i and v_j are merged.

2. If there are two distinct atoms $v_i \lhd v_j$ and $v_k \lhd v_j$ then all occurrences of v_i are replaced by v_k, i.e., the nodes v_i and v_k are merged.

This can create more cycles but it will always hold for each undirected cycle, i.e., a cycle that ignores the direction of the edges, that (p1) it contains only $\lhd$ and $\prec$ edges and (p2) is not a directed cycle, because these properties hold for the initial TDG and are preserved by the rules. Another property for which this holds is that (p3) if there is an $\prec$ edge between two nodes then there are two $\lhd$ edges that define a common parent. Because the rules are applied exhaustively it will also hold in the result that (p4) there are no two distinct atoms v_i and v_k for which there is a node v_j such that $v_i \lhd v_j$ and $v_k \lhd v_j$.

Finally, we check if there is a conflict, i.e., there are two atoms $a(v_i)$ and $b(v_i)$ with $a \neq b$. If so then this TDG is not satisfiable and because all the applied rules maintain satisfiability also the original TDG and consequently also the original path expression is not satisfiable.

If there is no such conflict then we can construct a satisfying XML tree from the obtained TDG as follows. We divide the variables into clusters which are maximal sets of variables that are directly or indirectly connected by $\lhd$ atoms. Note that because of properties p2 and p4 the $\lhd$ atoms define a tree over the variables in each cluster and because of p2 and p3 it is possible to complete the $\prec$ relationship for this tree to a strict total order that satisfies PTW1 and PTW2. Moreover, because of the properties p1 and p3 there can only be $\lhd^*$ and $\lhd^+$ edges between variables in different clusters and these edges will never define directed or undirected cycles over these clusters. Therefore we can sort the clusters topologically and connect the trees for each cluster by considering each cluster and its immediate successor in the topological sort (if there is one) and if there is an $\lhd^*$ or $\lhd^+$ edge from v_i in the first cluster to v_j in the second cluster then we add an $\lhd$ edge from v_i to the root of v_js cluster, if there is not an $\lhd^*$ or $\lhd^+$ edge between the clusters then we add an $\lhd$ edge between an arbitrary node in the first clusters with the root of the second cluster.

Finally, we have to complete the $\prec$ relationship, which was already completed for each cluster, for the complete tree. Since there are only $\lhd^+$ and $\lhd^*$ edges between the clusters and these define a tree over these clusters, it follows that we can complete the $\prec$ relationship to a strict total order over all the nodes that satisfies PTW1 and PTW2. □

Theorem 7.3. *Deciding satisfiability for $\mathcal{P}_{\Uparrow}$ is in PTIME.*

Proof. (Sketch) This proof proceeds similar to the proof of Theorem 7.2, but now we also merge v_i and v_r if there is an atom $v_i \lhd^* v_r$. Furthermore we also check for conflicts in the form of atoms $v_i \prec v_r$, $v_i \lhd v_r$ and $v_i \lhd^+ v_r$. Finally, we attempt to construct a satisfying XML tree in the same way except when there is an $\lhd^*$ or $\lhd^+$ edge that arrives in the cluster that contains the root variable v_r. Note that if we follow the procedure of the previous proof then this root node would become the child of another node, which is not allowed in an XML tree. Therefore we do here the following. Let the atom in question be $v_i \lhd^+ v_j$ or $v_i \lhd^* v_j$. Then we attempt to merge v_i and its ancestors (as defined by the $\lhd$ atoms) with an ancestor of v_j and its ancestors. If this is possible without a conflict between their tag names and without introducing a parent of the root then we merge them such that v_i is merged with the lowest possible ancestor of v_j in its cluster. This is repeated until the cluster with v_r has no more incoming $\lhd^*$ and $\lhd^+$ edges.

If by then we still have not found a conflict then we can proceed to construct the satisfying XML tree as in the previous proof by making sure that the cluster with v_r becomes the smallest cluster. If we do find a conflict then it is not possible to avoid it by merging v_i with a higher ancestor because that would only limit the possibilities more for subsequent merges. This is because it holds that the cluster with v_r has always just one incoming $\lhd^*$ or $\lhd^+$ edge unless the original path expression used $\Uparrow$ in other places then the beginning of the path. However, in the latter case we can decide satisfiability by splitting the path at the intermediate $\Uparrow$ step and deciding it separately for the two resulting path expressions. □

8 Summary and Discussion

For tree description graphs the problem of deciding satisfiability was shown to be NP complete. This result is similar to that in [10] except that they require atoms of the form $v_0 : f(v_1, \ldots, v_n)$ that specify that v_0 is labelled with f and whose set of children is exactly $\{v_1, \ldots, v_n\}$. Our result shows that even with only unary atoms ($n = 1$) the problem is already NP hard.

For fragments of XPath the complexity results are given by the following table.

$\Uparrow$	[]	$\cup$	$\cap$	$-$	**Complexity**
•					PTIME
	•				PTIME
			•		NP-complete
•	•				NP-complete
	•	•			NP-complete
•	•	•	•		NP-complete
				•	NP-hard

Remaining open problems are finding a better lower bound and an upper bound for $\mathcal{P}_-$ and classifying the fragments $\mathcal{P}_\cup$ and $\mathcal{P}_{\Uparrow,\cup}$.

Another open problem is the relationship between $\mathcal{P}_{\Uparrow,[\,],\cap}$ and tree description graphs. As was shown by Theorem 4.1 every path expression in this fragment can be translated to an equivalent TDG, but whether the converse holds is still unknown.

Finally, given the research that has been done on the containment problem for XPath expressions given a DTD [6, 13, 17] which limits itself mainly to XPath fragment with forward axes, and the results in this paper that seem to indicate that the satisfiability problem is sometimes simpeler, even if also reverse axes are allowed, it will be interesting to see what the complexity of the satisfiability problem is in the context of DTDs. Although some algorithms have been suggested such as in [11] this is still largely unknown.

Acknowledgement. I would like to thank the anonymous referees for their corrections, comments and suggestions for improvement.

References

[1] Sihem Amer-Yahia, SungRan Cho, Laks V. S. Lakshmanan, and Divesh Srivastava. Minimization of tree pattern queries. In *SIGMOD Conference*, 2001.

[2] Michael Benedikt, Wenfei Fan, and Gabriel M. Kuper. Structural properties of XPath fragments. In Diego Calvanese, Maurizio Lenzerini, and Rajeev Motwani, editors, *Proceedings of the 9th International Conference on Database Theory (ICDT 2003)*, volume 2572 of *LNCS*, pages 79–95. Springer, 2003.

[3] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernandez, Michael Kay, Jonathan Robie, and Jérôme Siméon. XML path language (XPath) 2.0. W3C Working Draft, 2002. `http://www.w3.org/TR/xpath20/`.

[4] Manuel Bodirsky and Martin Kutz. Pure dominance constraints. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2002)*, 2002.

[5] Tom Cornell. On determining the consistency of partial descriptions of trees. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, pages 163–170. Morgan Kaufmann, 1994.

[6] Alin Deutsch and Val Tannen. Containment and integrity constraints for xpath fragments. In *Proceedings of the 8th International Workshop on Knowledge Representation Meets Databases (KRDB 2001)*, 2001.

[7] Denise Draper, Peter Fankhauser, Mary Fernández, Ashok Malhotra, Kristoffer Rose, Michael Rys, Jérôme Siméon, and Philip Wadler. XQuery 1.0 and XPath 2.0 formal semantics, 2002. `http://www.w3.org/TR/xquery-semantics/`.

[8] Mary Fernández, Ashok Malhotra, Jonathan Marsh, Marton Nagy, and Norman Walsh. XQuery 1.0 and XPath 2.0 data model. W3C Working Draft, 2002. `http://www.w3.org/TR/xpath-datamodel/`.

[9] Michael R. Garey and David S. Johnson. *Computers and Intractability – A guide to NP-completeness*. W. H. Freeman and Company, San Francisco, 1979.

[10] Alexander Koller, Joachim Niehren, and Ralf Treinen. Dominance constraints: Algorithms and complexity. In *Proceedings of the Third Conference on Logical Aspects of Computational Linguistics*, volume 2014 of *Lecture Notes in Computer Science*, pages 106–125, Grenoble, 2001. Springer - Verlag.

[11] April Kwong and Michael Gertz. Schema-based optimization of xpath expressions. Technical report, University of California at Davis, 2002.

[12] Gerome Miklau and Dan Suciu. Containment and equivalence for an XPath fragment. In *Symposium on Principles of Database Systems*, pages 65–76, 2002.

[13] Frank Neven and Thomas Schwentick. XPath containment in the presence of disjunction, DTDs and variables. In Diego Calvanese, Maurizio Lenzerini, and Rajeev Motwani, editors, *Proceedings of the 9th International Conference on Database Theory (ICDT 2003)*, volume 2572 of *LNCS*, pages 315–329. Springer, 2003.

[14] James Rogers and K. Vijay-Shanker. Reasoning with descriptions of trees. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 72–80, 1992.

[15] Philip Wadler. Two semantics for XPath, 1999. `http://www.cs.bell-labs.com/who/wadler/topics/xml.html`.

[16] Peter T. Wood. Minimising simple XPath expressions. In *Proceedings of the Fourth International Workshpo on the Web and Databases (WebDB 2001)*, pages 13–18, Santa Barbara, California, May 2001.

[17] Peter T. Wood. Containment for XPath fragments under DTD constraints. In Diego Calvanese, Maurizio Lenzerini, and Rajeev Motwani, editors, *Proceedings of the 9th International Conference on Database Theory (ICDT 2003)*, volume 2572 of *LNCS*, pages 300–314. Springer, 2003.