Containment for XPath Fragments under DTD Constraints

Peter T. Wood

Birkbeck College, University of London

Abstract. The containment and equivalence problems for various fragments of XPath have been studied by a number of authors. For some fragments, deciding containment (and even minimisation) has been shown to be in PTIME, while for minor extensions containment has been shown to be CONP-complete. When containment is with respect to trees satisfying a set of constraints (such as a schema or DTD), the problem seems to be more difficult. For example, containment under DTDs is CONP-complete for an XPath fragment denoted $\mathrm{XP}^{\{[\,]\}}$ for which containment is in PTIME. It is also undecidable for a larger class of XPath queries when the constraints are so-called simple XPath integrity constraints (SXICs). In this paper, we show that containment is decidable for an important fragment of XPath, denoted $\mathrm{XP}^{\{[\,],*,//\}}$, when the constraints are DTDs. We also identify XPath fragments for which containment under DTDs can be decided in PTIME.

1 Introduction

XPath [17] is a language for selecting nodes from XML document trees, and as such plays a crucial role in other XML technologies such as XSLT [18] and XQuery [19]. The expressions of XPath can be interpreted as simple queries on tree structures. The *containment problem* for XPath is to decide, given two XPath queries Q_1 and Q_2, whether Q_1 *contains* Q_2; that is, for every XML tree t, whether the output of Q_1 on t contains the output of Q_2 on t. Since XPath query containment has many applications in XML querying, integration, transformation and active rule [2] environments, it has been the subject of much study recently [1,7,9,10,15,16].

Most of the papers cited above have studied different fragments of XPath, denoted $\mathrm{XP}^{\{[\,]\}}$, $\mathrm{XP}^{\{[\,],*\}}$, $\mathrm{XP}^{\{[\,],//\}}$ and $\mathrm{XP}^{\{[\,],*,//\}}$ in [9], depending on which XPath constructs are included in the fragment. All these fragments include node tests, composition of location steps (/), and predicates ([ ]). $\mathrm{XP}^{\{[\,],*\}}$ adds wildcards ($*$), $\mathrm{XP}^{\{[\,],//\}}$ adds the descendant axis (//), and $\mathrm{XP}^{\{[\,],*,//\}}$ includes both of these. Although not utilising the full capabilities of XPath, these fragments are commonly used to select nodes in XSLT, for example, and are interesting because of the relative complexity of the containment problem for them.

Example 1. The XPath query $a//b[*/i]/g$ selects nodes labelled with g (which we call g-nodes for short) that are children of b-nodes, such that the b-nodes are both descendants of the root a-node and have an i-node as a grandchild.

It is also useful to be able to view a query in $XP^{\{[\,],*,//\}}$ as a *tree pattern* [1, 9]. Fig. 1 gives the tree pattern for the query $a//b[*/i]/g$. In a tree pattern, double-lines indicate *descendant* edges (corresponding to //), while the return node is indicated in bold.

When using the descendant axis at the beginning of a predicate, we need to include "." (denoting the context node) in order to state that we want to search for descendants from that position in the query; without the "." searching would start from the root node. In fact, the XPath fragments we consider do not allow the second form of query. □

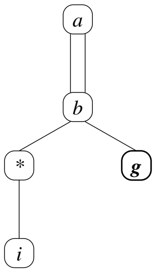

Fig. 1. The tree pattern for the XPath query $a//b[*/i]/g$.

Deciding containment for $XP^{\{[\,],*\}}$ is in PTIME [16], a result which follows from the PTIME containment for acyclic conjunctive queries [20] (as pointed out in [9]). Containment for $XP^{\{[\,],//\}}$ is also in PTIME [1]. However, as shown recently by Miklau and Suciu [9], when we extend the XPath fragment to $XP^{\{[\,],*,//\}}$, containment becomes CONP-complete.

We are interested in studying containment in the presence of Document Type Definitions (DTDs). DTDs provide a means for typing, and therefore constraining, the structure of XML documents. Hence, while query Q_1 may not contain query Q_2 in general, it may be the case that, given a DTD D, Q_1 contains Q_2 when both are evaluated on documents valid for D.

Example 2. For the queries $Q_1 = a[b]/c$ and $Q_2 = a/c$, it is obvious that $Q_2 \supseteq Q_1$. The converse, however, does not hold since on a tree with only an a-node with c-node as a child, Q_2 returns the c-node while Q_1 returns the empty set. Now consider the following DTD D

$$\begin{aligned} a &\to (b, ((b, c)|d)) \\ b &\to ((e|f), (g|h)) \\ e &\to (i) \\ f &\to (i) \end{aligned}$$

which defines XML document trees that have a root node labelled a. The DTD D states, for example, that every node labelled a in a document tree valid for D

(or which *satisfies* D) must have a node labelled b as a child. This is an example of what has been called a *child constraint* [15]. Such a constraint allows us to infer that Q_1 contains Q_2 under D, written $Q_1 \supseteq_{SAT(D)} Q_2$.

D also requires that every b-node must have an i-node as a descendant. This means that $a[.//i]/c \supseteq_{SAT(D)} a/c$. As a final example, D ensures that every path from an a-node to an i node must pass through a b-node, which means that $a//b//i \supseteq_{SAT(D)} a//i$. □

As one might expect, when we consider containment of queries under constraints, the complexity increases. Given a DTD D, deciding containment under D (*D-containment* for short), even for queries in $\mathrm{XP}^{\{[\,]\}}$, is CONP-complete [10, 16]. Containment is undecidable when the XPath fragment includes $\mathrm{XP}^{\{[\,],*,//\}}$ along with disjunction, variable binding and equality testing, and the constraints include so-called bounded simple XPath integrity constraints (SXICs) and those (unbounded) constraints implied by DTDs [7]. The decidability of containment of this XPath fragment in the presence of DTDs was stated as an open problem in [7]. In this paper, we prove that D-containment is decidable (EXPTIME-complete, in fact) for $\mathrm{XP}^{\{[\,],*,//\}}$. The same result is proved independently in [10].

The intractability of D-containment for $\mathrm{XP}^{\{[\,]\}}$ comes from the fact that inferring even some simple contraints from a DTD seems to be intractable [16]. As a result, we have been investigating classes of constraints which are implied by a DTD D and which are both necessary and sufficient to determine D-containment for various classes of XPath queries in PTIME. For example, it was shown in [16] that so-called *sibling constraints* capture D-containment for queries in $\mathrm{XP}^{\{[\,]\}}$ that are *duplicate-free.* A query Q in $\mathrm{XP}^{\{[\,]\}}$ is *duplicate-free* if each node in the tree pattern corresponding to Q has children with distinct labels.

In this paper, we show that if DTD D is *duplicate-free*, then D-containment for $\mathrm{XP}^{\{[\,]\}}$ is captured by sibling constraints and *functional constraints*, and can be decided in PTIME. A DTD D is *duplicate-free* if each element name n appears at most once in each content model (this does not preclude a node n having multiple children with the same label in a tree satisfying D since n may still have a closure operator applied to it).

Example 3. Consider the following DTD D which is duplicate-free:

$$\begin{aligned} a &\rightarrow ((b*, c) \mid d) \\ b &\rightarrow (e \mid f) \\ c &\rightarrow (g?, h?) \end{aligned}$$

D does not imply any child constraints. However, it is the case that if an a-node has a b-node as a child, then it has a c-node as a child. This is an example of a *sibling constraint* (SC) [16], a generalisation of a child constraint, which allows us to show that $a[b][c] \supseteq_{SAT(D)} a[b]$.

It is also the case that D requires that every a-node has at most one c-node as a child. This *functional constraint* (FC) allows us to show that $a/c[g] \supseteq_{SAT(D)} a[c/g]/c$. □

It is claimed in [16] that, if a DTD D is duplicate-free, we can rewrite an $\mathrm{XP}^{\{[\,]\}}$ query that is not duplicate-free as one that is duplicate-free while maintaining D-equivalence. However, the duplicate-free DTD D of Example 3 and query $Q = a[b/e][b/f]$ show this is not true. Q is D-satisfiable but there is no duplicate-free XPath query equivalent to Q. Hence, we need the results in the present paper to show PTIME containment under duplicate-free DTDs for $\mathrm{XP}^{\{[\,]\}}$.

Finally, given the examples of constraints implied by a DTD in Example 2, one might wonder whether a richer class of constraints than SCs and FCs, but less powerful than DTDs themselves, might capture D-containment for $\mathrm{XP}^{\{[\,]\}}$ when neither D nor the queries are duplicate-free. We prove that this is not the case, in the sense that to capture D-containment for $\mathrm{XP}^{\{[\,]\}}$ requires the ability to express exactly the *unordered* language generated by D.

In the next section, we introduce the necessary definitions and notation for XML trees, XPath queries and DTDs. In Section 3, we prove that containment under DTDs is decidable for queries in $\mathrm{XP}^{\{[\,],*,//\}}$. We show this by transforming queries in $\mathrm{XP}^{\{[\,],*,//\}}$ to regular tree grammars and using decidability and closure results for regular tree grammars, since DTDs are a special case of regular tree grammars. In Section 4, we show that, for queries in $\mathrm{XP}^{\{[\,]\}}$, sibling and functional constraints are necessary and sufficient to decide containment under duplicate-free DTDs in PTIME. We prove in Section 5 that this result cannot be extended beyond duplicate-free DTDs. Section 6 contains a discussion of related work, while Section 7 concludes.

2 XML Trees, XPath Queries, and DTDs

Let Σ be a finite alphabet of XML element names. Since the XPath fragments we consider cannot query attributes or the textual contents of nodes, we define a *document tree* (or simply *tree*) over Σ to be an ordered, unranked finite structure with nodes labelled by element names from Σ. So all leaf nodes (elements) are assumed to be empty. The set of all trees over Σ is denoted by T_Σ. Given a tree $t \in T_\Sigma$, the root of t is denoted by $root(t)$, the nodes of t by $nodes(t)$, and the label of node $x \in nodes(t)$ by $\lambda(x) \in \Sigma$.

Expressions in $\mathrm{XP}^{\{[\,],*,//\}}$ (which we also loosely refer to as queries) are defined by the following grammar, in which n denotes an element name and "." refers to the current node:

$$p \rightarrow p\ \text{'/'}\ p \mid p\ \text{'//'}\ p \mid p\ \text{'['}\ p\ \text{']'} \mid n \mid \text{'*'} \mid \text{'.'}$$

Expressions enclosed in '[' and ']' are called *predicates*. The use of the current node in expressions such as $a/./b$ or $a//.//b$, it can be eliminated. Expressions in $\mathrm{XP}^{\{[\,]\}}$ do not use ".", "*" or "//".

Given a query Q in $\mathrm{XP}^{\{[\,],*,//\}}$ and a tree $t \in T_\Sigma$, $Q(t)$ denotes the set of nodes that is the result of evaluating Q on t according to the semantics given in [14]. We assume that the context node for evaluating Q on t is always $root(t)$. If $Q(t) \neq \emptyset$, we say that t *satisfies* Q. The set of trees over Σ satisfying Q is denoted $SAT_\Sigma(Q)$. Let P and Q be queries in $\mathrm{XP}^{\{[\,],*,//\}}$. We say that P

contains Q, written $P \supseteq Q$, if for all trees $t \in T_\Sigma$, $P(t) \supseteq Q(t)$. In addition, P is *equivalent* to Q, written $P \equiv Q$, if $P \supseteq Q$ and $Q \supseteq P$.

Similar to [1,9], we consider queries also as *tree patterns*. A pattern Q is an unordered tree over alphabet $\Sigma \cup \{*\}$ with a distinguished subset of edges called *descendant edges*, and a distinguished node called the *result node*. Edges that are not descendant edges are called *child edges*. An example of a tree pattern is given in Fig. 1, where child edges are represented by single lines, descendant edges by double lines, and the result node by a boldface circle and label. The result node of pattern Q is denoted $\mathit{result}(Q)$. From now on, we use XPath queries and tree patterns interchangeably.

As in [9], an *embedding* or *homomorphism* from pattern Q to tree t is a mapping h from the nodes of Q to the nodes of t such that (1) h maps $\mathit{root}(Q)$ to the $\mathit{root}(t)$, (2) for each node x in Q, $\lambda(x) = *$ or $\lambda(x) = \lambda(h(x))$, and (3) for each node x and y in Q, if (x, y) is a child edge in Q then $(h(x), h(y))$ is an edge in t, and if (x, y) is a descendant edge in Q then $h(y)$ is a proper descendant of $h(x)$ in t. If the result node of Q is x, then

$$Q(t) = \{(h(x)) \mid h \text{ is a homomorphism from } Q \text{ to } t\}$$

$Q(t)$ is represented as a set of tuples rather than simply a set of elements, because we also want to allow Boolean patterns and to be able to distinguish between Boolean patterns that are satisfied by some tree and those that are not [9]. A *Boolean* pattern Q is a pattern in which there is no result node: $Q(t)$ evaluates to the empty tuple if there a homomorphism from Q to t; otherwise, $Q(t) = \emptyset$. When Q is not a Boolean pattern, we will often consider $Q(t)$ to be simply a set of elements.

A containment mapping between queries is similar to a homomorphism from a query to a tree, the differences arising because queries have result nodes and descendant edges which trees do not. A *containment mapping* from query Q_1 to query Q_2 is a mapping h from the nodes of Q_1 to the nodes of Q_2 such that (1) h maps $\mathit{root}(Q_1)$ to $\mathit{root}(Q_2)$, (2) h maps $\mathit{result}(Q_1)$ to $\mathit{result}(Q_2)$, (3) for each node x in Q_1, $\lambda(x) = *$ or $\lambda(x) = \lambda(h(x))$, and (4) for each node x and y in Q_1, if (x, y) is a child edge in Q_1 then $(h(x), h(y))$ is a child edge in Q_2, and if (x, y) is a descendant edge in Q_1 then $(h(x), \ldots, h(y))$ is a path of child and/or descendant edges in Q_2.

The existence of a containment mapping is always a sufficient condition for containment between queries. It is also a necessary condition for containment for $\mathrm{XP}^{\{[\,],*\}}$ and $\mathrm{XP}^{\{[\,],//\}}$ (and, of course, for $\mathrm{XP}^{\{[\,]\}}$), but it is *not* a necessary condition for containment for $\mathrm{XP}^{\{[\,],*,//\}}$ [9].

A *document type definition* (DTD) D over finite alphabet Σ consists of a root type in Σ, denoted $\mathit{root}(D)$, and a mapping that associates with each $a \in \Sigma$ a regular expression over Σ. If the mapping associates with a the regular expression R^a, then we say that R is the *content model* for a and write $a \to R^a$ (which we also call a *production*). By convention, we often simply write down the productions for D, with the first production being for the root type and with Σ comprising all symbols which appear in the productions. For example, the root type of the DTD D of Example 2 is taken as a and $\Sigma = \{a, b, c, d, e, f, g, h, i\}$.

For regular expressions, we use the conventions from XML DTDs, namely that ‘,’ denotes concatenation, ‘|’ denotes alternation, and ‘$*$’ denotes Kleene closure. We represent the regular expression denoting the empty string by ϵ, so that, for regular expression R, R^+ is shorthand for R, R^*, and $(R)?$ is shorthand for $(R \mid \epsilon)$. If R is a regular expression, then $L(R)$ is the *language* denoted by R.

A tree $t \in T_\Sigma$ *satisfies* a DTD D over Σ if $\lambda(root(t)) = root(D)$ and for each node x in t with sequence of children $y_1, \ldots, y_n$, the string $\lambda(y_1) \cdots \lambda(y_n)$ is in $L(R^{\lambda(x)})$ (the language for the content model of the label of x). The set of trees satisfying DTD D is denoted $SAT(D)$.

Given XPath queries Q_1 and Q_2, Q_1 *D-contains* Q_2, written $Q_1 \supseteq_{SAT(D)} Q_2$, if, for every tree $t \in SAT(D)$, $Q_1(T) \supseteq Q_2(T)$. Q_1 and Q_2 are *D-equivalent*, written $Q_1 \equiv_{SAT(D)} Q_2$, if $Q_1 \supseteq_{SAT(D)} Q_2$ and $Q_2 \supseteq_{SAT(D)} Q_1$.

Let $t \in T_\Sigma$ be a (document) tree, $a, c \in \Sigma$ be element names, and $B \subseteq \Sigma$ be a set of element names. Tree t *satisfies* the *sibling constraint* (SC) $a : B \Downarrow c$ if whenever a node labelled a in t has children labelled with each $b \in B$, it has a child node labelled with c [16]. When $B = \emptyset$, the SC is called a *child constraint* [15]. An SC (over Σ) is *trivial* if it is satisfied by every tree $t \in T_\Sigma$. So $a : B \Downarrow c$ is trivial if $c \in B$. If S is a set of SCs over Σ, then $SAT(S)$ denotes the set of trees in T_Σ which satisfy each SC in S.

Let s be the SC $a : B \Downarrow c$. Regular expression R^a *implies* the SC s, written $R^a \models s$ if whenever a string $w \in L(R^a)$ contains each element of B, it also contains c. SC implication was shown to be CONP-hard in [16]. DTD D *implies* the SC s, written $D \models s$, if R^a implies s or, equivalently, each $t \in SAT(D)$ satisfies s. D *implies* the set of SCs S, written $D \models S$, if $SAT(D) \subseteq SAT(S)$.

Let $t \in T_\Sigma$ and $a, b \in \Sigma$ be element names. Tree t *satisfies* the *functional constraint* (FC) $a \downarrow b$ if no node labelled a in t has two distinct children labelled with b. If C is a set of SCs and FCs over Σ, then $SAT(C)$ denotes the set of trees in T_Σ which satisfy each SC and FC in C. The definitions for a regular expression and a DTD implying an FC, or a set of FCs, are analogous to those for SCs.

3 Decidability of Containment under DTDs

When the XPath fragment includes $\text{XP}^{\{[\,],*,//\}}$ along with disjunction, variable binding and equality testing, Deutsch and Tannen state that the decidability of containment in the presence of DTDs is an open problem [7]. In this section, we contribute to this investigation by showing that containment of $\text{XP}^{\{[\,],*,//\}}$ queries under DTDs is decidable. (In fact, including disjunction in the XPath fragment as well does not change the decidability result.) We do this by showing that, given a query Q in $\text{XP}^{\{[\,],*,//\}}$ and an alphabet Σ for a DTD D, we can construct a regular tree grammar (RTG) G such that the set of trees generated by the grammar is precisely the set of trees that satisfy Q. The result then follows from the facts that DTDs are a special case of RTGs, that RTGs are closed under intersection [6], and that containment is decidable (and EXPTIME-complete) for RTGs [4,12,13]. This same result is proved independently in [10].

The definition of regular tree grammars we use is similar to that in [6]. A *regular tree grammar* (RTG) G is a 4-tuple $\langle \Sigma, N, P, n_0 \rangle$, where

1. Σ is a finite set of element names,
2. N is a finite set of nonterminals,
3. P is a finite set of productions of the form $n \to a(R)$, where $n \in N$, $a \in \Sigma$, and R is a regular expression over N,
4. $n_0 \in N$ is the start symbol.

RTG G *allows* a document tree t if t can be derived from n_0 by applying productions from P and t does not contain any nonterminals. A production $n \to a(R)$ is applied to a tree t by replacing the nonterminal n in t by a tree $a(t_1, \ldots, t_k)$, where $t_1 \cdots t_k \in L(R)$. The *regular tree language* $L(G)$ is the set of document trees allowed by the grammar G. Given two RTGs G_1 and G_2, we write $G_1 \supseteq G_2$ if $L(G_1) \supseteq L(G_2)$.

Given a query Q in $\mathrm{XP}^{\{[\,],*,//\}}$, we want to derive an RTG G such that $L(G) = SAT_\Sigma(Q)$, the set of trees over Σ satisfying Q. We can use this to decide D-containment for queries Q_1 and Q_2 as described below.

Firstly, Miklau and Suciu show that there is a translation from $\mathrm{XP}^{\{[\,],*,//\}}$ queries over Σ to Boolean $\mathrm{XP}^{\{[\,],*,//\}}$ queries over an extended alphabet Σ' such that for any $\mathrm{XP}^{\{[\,],*,//\}}$ queries Q_1 and Q_2 and their translations Q_1' and Q_2', $Q_1 \supseteq Q_2$ if and only if $Q_1' \supseteq Q_2'$ [9]. Now let G_1 and G_2 be RTGs such that $L(G_1) = SAT_{\Sigma'}(Q_1')$ and $L(G_2) = SAT_{\Sigma'}(Q_2')$. Then $Q_1' \supseteq_{SAT(D)} Q_2'$ if and only if $D \cap G_1 \supseteq D \cap G_2$. So for the rest of this section, we assume that all queries are Boolean.

Let alphabet $\Sigma = \{a_1, \ldots a_k\}$. As a shorthand notation, we write

$$n \to \Sigma\ (r)$$

for the set of productions

$$\begin{gathered} n \to a_1\ (r) \\ \vdots \\ n \to a_k\ (r) \end{gathered}$$

In order to generate an arbitrary tree over Σ, we define a nonterminal n_Σ by the (shorthand) production:

$$n_\Sigma \to \Sigma\ (n_\Sigma^*)$$

Example 4. Given query $Q = a/b$ over alphabet Σ, the productions for the RTG corresponding to Q are

$$\begin{gathered} n_a \to a\ (n_\Sigma^*\ n_b\ n_\Sigma^*) \\ n_b \to b\ (n_\Sigma^*) \end{gathered}$$

since Q matches trees in which nodes labelled a can have arbitrary children labelled with elements from Σ that precede and/or follow the child labelled b, which in turn can be the root of an arbitrary subtree. $\square$

We also have to take into account that order in RTGs is significant. So for a query which has a node with more than one child, we need to represent all possible orderings of the children in the RTG. To make this more manageable, we use the & operator from SGML as a shorthand notation: for all pairs of symbols a and b in Σ, $a \,\&\, b \equiv ((a, b) \mid (b, a))$. Descendant edges in queries are modelled by recursive productions in the RTG.

Example 5. The productions for query $a[b][c]$ are

$$\begin{aligned} n_a &\to a\ (n_\Sigma^* \,\&\, n_b \,\&\, n_\Sigma^* \,\&\, n_c \,\&\, n_\Sigma^*) \\ n_b &\to b\ (n_\Sigma^*) \\ n_c &\to c\ (n_\Sigma^*) \end{aligned}$$

while those for query $a//b$ are

$$\begin{aligned} n_a &\to a\ (n_\Sigma^*\ n_b\ n_\Sigma^*) \\ n_b &\to b\ (n_\Sigma^*) \\ n_b &\to \Sigma\ (n_\Sigma^*\ n_b\ n_\Sigma^*) \end{aligned}$$

□

Given an alphabet $\Sigma = \{a_1, \dots a_k, *\}$ and a query Q in $\mathrm{XP}^{\{[\,],*,//\}}$ with m nodes, we construct an RTG G from Q as follows. First number each node in Q uniquely, with the root node numbered 1. The RTG G *corresponding to* Q is given by $\langle \Sigma, N, P, n_1 \rangle$, where $N = \{n_1, \dots, n_m, n_\Sigma\}$, and P is constructed inductively as follows.

1. If node i in Q is a leaf node, then P includes
$$n_i \to a_j\ (n_\Sigma^*)$$
if i has label $a_j \in \Sigma$, or
$$n_i \to \Sigma\ (n_\Sigma^*)$$
if i has label $*$.
2. If node i in Q has child nodes $j_1, \dots, j_m$, then P includes
$$n_i \to a_l\ (n_\Sigma^* \,\&\, n_{j_1} \,\&\, n_\Sigma^* \,\&\cdots\,\&\, n_\Sigma^* \,\&\, n_{j_m} \,\&\, n_\Sigma^*)$$
if i has label $a_l \in \Sigma$, or
$$n_i \to \Sigma\ (n_\Sigma^* \,\&\, n_{j_1} \,\&\, n_\Sigma^* \,\&\cdots\,\&\, n_\Sigma^* \,\&\, n_{j_m} \,\&\, n_\Sigma^*)$$
if i has label $*$.
3. If node i in Q is connected to its parent by a descendant edge, then P includes
$$n_i \to \Sigma\ (n_\Sigma^*\ n_i\ n_\Sigma^*)$$

We now define the notion of a node-disjoint embedding, which is useful when defining the relationship between query Q and the trees generated by the RTG corresponding to Q. A *node-disjoint embedding* from query Q to tree t is a pair $h = (h_1, h_2)$, where h_1 is a one-to-one mapping from the nodes of Q to the nodes of t such that (1) h_1 maps $root(Q)$ to $root(t)$, and (2) for each node x in Q, $\lambda(x) = *$ or $\lambda(x) = \lambda(h_1(x))$, and h_2 is a one-to-one mapping from the edges of Q to paths of t such that (3) for each node x and y in Q, if $e = (x, y)$ is a child edge in Q, then $h_2(e) = (h_1(x), h_1(y))$ is an edge in t, (4) for each node x and y in Q, if $e = (x, y)$ is a descendant edge in Q, then $h_2(e) = (h_1(x), \ldots, h_1(y))$ is a path in t. We also require that all paths in the image of h_2 are node-disjoint, except that two paths may have the same initial node. It follows immediately from the definition that, if $h = (h_1, h_2)$ is a node-disjoint embedding from query Q to tree t, then h_1 is an embedding from Q to t.

Lemma 1. *Let Q be a query over Σ in $XP^{\{[\,],*,//\}}$, G be the RTG corresponding to Q, and t be a tree over Σ. There is a node-disjoint embedding from Q to t if and only if $t \in L(G)$.*

A *contraction* of a query Q_1 is a query Q_2 comprising a subset of the nodes of Q_1 such that there is a containment mapping from Q_1 to Q_2. Fig. 2 gives an example of some contractions for a query in $\text{XP}^{\{[\,],*,//\}}$.

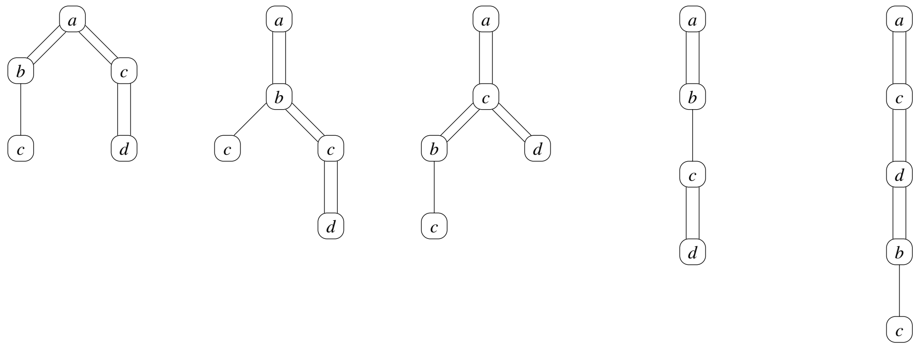

Fig. 2. Some contractions for the XPath query $a[.//b/c][.//c//d]$.

Lemma 2. *Let Q_1 be a query over Σ in $XP^{\{[\,],*,//\}}$ and t be a tree over Σ. If there is an embedding from Q_1 to t, then there is a contraction Q_2 of Q_1 such that there is a node-disjoint embedding from Q_2 to t.*

It is straightforward to construct an RTG G from a set of contractions for a query Q. Given a set C of contractions for Q, we number the nodes of queries uniquely throughout C, except that the root node for each contraction is numbered 1. The set of productions of G is then simply the union of the sets of productions for each of the individual contractions in C.

Lemma 3. *Let Q be a query over Σ in $XP^{\{[\,],*,//\}}$ and G be the RTG corresponding to the set of contractions of Q. Then $L(G) = SAT_{\Sigma}(Q)$.*

Example 6. Consider the following pair of queries $Q_1 = a/*//b$ and $Q_2 = a//*/b$ from [9]. Although Q_1 and Q_2 are equivalent, there is no containment mapping between them. Given alphabet Σ, the RTG productions for Q_1 and Q_2 are

$$\begin{array}{lcl} n_a \rightarrow a\ (n_\Sigma^* \ n_* \ n_\Sigma^*) & & n_a \rightarrow a\ (n_\Sigma^* \ n_* \ n_\Sigma^*) \\ n_* \rightarrow \Sigma\ (n_\Sigma^* \ n_b \ n_\Sigma^*) & \text{and} & n_* \rightarrow \Sigma\ (n_\Sigma^* \ n_* \ n_\Sigma^*) \\ n_b \rightarrow \Sigma\ (n_\Sigma^* \ n_b \ n_\Sigma^*) & & n_* \rightarrow \Sigma\ (n_\Sigma^* \ n_b \ n_\Sigma^*) \\ n_b \rightarrow b\ (n_\Sigma^*) & & n_b \rightarrow b\ (n_\Sigma^*) \end{array}$$

respectively, Clearly, the above two sets of productions are equivalent. □

Theorem 1. *Let D be a DTD over Σ, Q_1 and Q_2 be queries over Σ in $XP^{\{[\,],*,//\}}$, and G_1 and G_2 be the RTGs corresponding to the sets of contractions of Q_1 and Q_2, respectively. Then $Q_1 \supseteq_{SAT(D)} Q_2$ if and only if $D \cap G_1 \supseteq D \cap G_2$.*

Corollary 1. *Containment for queries in $XP^{\{[\,],*,//\}}$ under DTDs is decidable and, in fact,* EXPTIME-*complete [13].*

Example 7. Consider the DTD D from Example 2, along with queries $Q_1 = a[b[e][g]][d]$ and $Q_2 = a[b/e][b/g]][d]$. Note that Q_1 is a contraction of Q_2 and hence $Q_2 \supseteq Q_1$. Because D dictates that if an a-node has a d-child, then it has at most one b-child, we have that $Q_1 \supseteq_{SAT(D)} Q_2$.

The productions for G_2 corresponding to the set of contractions of Q_2 are

$$\begin{aligned} n_a &\rightarrow a\ (n_\Sigma^* \ \&\ n_{b_1} \ \&\ n_\Sigma^* \ \&\ n_{b_2} \ \&\ n_\Sigma^* \ \&\ n_d \ \&\ n_\Sigma^*) \\ n_{b_1} &\rightarrow b\ (n_\Sigma^* \ \&\ n_e \ \&\ n_\Sigma^*) \\ n_{b_2} &\rightarrow b\ (n_\Sigma^* \ \&\ n_g \ \&\ n_\Sigma^*) \\ n_a &\rightarrow a\ (n_\Sigma^* \ \&\ n_b \ \&\ n_\Sigma^* \ \&\ n_d \ \&\ n_\Sigma^*) \\ n_b &\rightarrow b\ (n_\Sigma^* \ \&\ n_e \ \&\ n_\Sigma^* \ \&\ n_g \ \&\ n_\Sigma^*) \\ n_d &\rightarrow d\ (n_\Sigma^*) \\ n_e &\rightarrow e\ (n_\Sigma^*) \\ n_g &\rightarrow g\ (n_\Sigma^*) \end{aligned}$$

while those for the RTG G_1, corresponding to the set of contractions of query Q_1, comprise the last 5 productions above.

DTD D can simply be represented as the following RTG, which we also denote by D

$$\begin{aligned} n_a &\rightarrow a\ ((n_b \ n_b \ n_c) \mid (n_d \ n_b)) \\ n_b &\rightarrow b\ ((n_e \mid n_f)\ (n_g \mid n_h)) \end{aligned}$$

along with productions of the form

$$n_x \to x\ (\epsilon)$$

for x instantiated to each of c, d, e, f, g and h. Now $D \cap G_1$ is

$$\begin{array}{ll} n_a \to a\ (n_d\ n_b) & n_d \to d\ (\epsilon) \\ n_b \to b\ (n_e\ n_g) & n_e \to e\ (\epsilon) \\ & n_g \to g\ (\epsilon) \end{array}$$

It turns out that $D \cap G_1 \equiv D \cap G_2$, since the first production for n_a in G_2 cannot be satisfied by any tree that satisfies D—it requires the existence of 2 children labelled with b and one with d. We conclude that $Q_1 \supseteq_{SAT(D)} Q_2$. □

4 PTIME Classes

We now turn our attention to fragments of XPath for which containment under DTDs can be tested in PTIME. We restrict ourselves to studying $\mathrm{XP}^{\{[\,]\}}$ and its subclasses, since it has been shown that, even for $\mathrm{XP}^{\{[\,]\}}$, deciding containment under DTDs in CONP-complete [10,16]. The approach adopted in [16], related to that in [7], is to look for classes of simple constraints implied by a DTD D which are necessary and sufficient to show D-containment. For example, it turns out that SCs are necessary and sufficient to show D-containment for *duplicate-free* queries in $\mathrm{XP}^{\{[\,]\}}$. Unfortunately, the CONP-hardness result relates to deciding whether a DTD implies an SC.

It is shown in [16] that when DTD D is *duplicate-free*, SC implication is in PTIME. In this section, we show that SCs and FCs are necessary and sufficient to decide D-containment for $\mathrm{XP}^{\{[\,]\}}$ queries when D is duplicate-free. We also show that finding the set of FCs implied by a DTD can be done in PTIME, as can deciding containment under duplicate-free DTDs for $\mathrm{XP}^{\{[\,]\}}$ queries.

Lemma 4. *Let D be a DTD, C be the set of SCs and FCs implied by D, and P and Q be $XP^{\{[\,]\}}$ queries. If $P \equiv_{SAT(C)} Q$, then $P \equiv_{SAT(D)} Q$.*

We can find the set of FCs implied by a DTD D over Σ in PTIME. For each regular expression R^a in D, we construct its NFA M^a and find the strongly-connected components (SCCs) of M^a. Then D implies the FC $a \downarrow b$ if and only if b labels a transition in M^a which is not part of any SCC. If $|\Sigma| = n$, at most n^2 FCs can be implied by D.

We are particularly interested in when SCs and FCs are *necessary* to show D-containment of $\mathrm{XP}^{\{[\,]\}}$ queries. For this, we need to consider duplicate-free DTDs. Recall that a DTD D is *duplicate-free* if in each regular expression R^a in D, each element name appears at most once in R^a.

We now consider some properties of duplicate-free DTDs. Firstly, for each regular expression R in a duplicate-free DTD D, we can construct an NFA M (with ϵ-transitions) accepting $L(R)$ such that no symbol labels more than one transition in M. This applies even if we use the operators + and ? in R. As a result of no symbol being repeated in M, each symbol in R appears either an unbounded number of times in strings in $L(R)$ or else it appears at most once in any string in $L(R)$.

Lemma 5. *Let D be a duplicate-free DTD. If in every tree in $SAT(D)$ whenever a node with label a has children with labels $b_1, \ldots, b_n$ it has a child with label c, then $D \models a : \{b_1, \ldots, b_n\} \Downarrow c$.*

Example 8. The result of Lemma 5 does not hold necessarily hold for DTDs which are not duplicate-free. Consider the DTD D from Example 2. Every tree in $SAT(D)$ with a node labelled a that has two children labelled b also has a child labelled c. This means, for example, that

$$a[b/e][b/f][c] \supseteq_{SAT(D)} a[b/e][b/f]$$

because the two copies of b in each of these expressions must always map to distinct nodes in any document tree (we have to consult D in order to determine this). However, the only (nontrivial) SC for a implied by D is $a : \emptyset \Downarrow b$. □

In order to test C-containment, for a set C of SCs and FCs, we introduce a variation of the *chase* [8], a procedure for applying a set of SCs and FCs to a query in $\text{XP}^{\{[\,]\}}$. Let P be a query in $\text{XP}^{\{[\,]\}}$ and C be the set of SCs and FCs implied by a DTD D.

1. Let $s \in C$ be a nontrivial SC of the form $a : B \Downarrow c$, where $B = \{b_1, \ldots, b_n\}$. Let u be a node in P with children $v_1, \ldots, v_n$ such that $\lambda(u) = a$, $\lambda(v_i) = b_i$, $1 \leq i \leq n$, and u does not have a child labelled c. The *result of applying* the SC s to u in P is a query which has the same nodes and edges as P and in addition has a child of u labelled c.
2. Let $f \in C$ be an FC of the form $a \downarrow b$. Let u be a node in P with distinct children v and w such that $\lambda(u) = a$ and $\lambda(v) = \lambda(w) = b$. The *result of applying* the FC f to u in P is a query $\theta(P)$, where θ maps v to w and is the identity elsewhere.

A *chasing sequence* of P by C is a sequence $P = P_0, \ldots, P_k$ such that for each $0 \leq i \leq k-1$, P_{i+1} is the result of applying some SC or FC in C to P_i, and no SC or FC can be applied to P_k. Note that the chasing sequence is finite. The *chase* of P by C, denoted $chase_C(P)$, is P_k.

For the set C of SCs and FCs implied by a DTD D and a query Q in $\text{XP}^{\{[\,]\}}$, $chase_C(Q)$ does not necessarily satisfy D. One reason is that D might require that one of two child nodes be present which cannot be captured by C. However, we do have the following.

Lemma 6. *Let D be a duplicate-free DTD, Q be an XPath query, and C be the set of SCs and FCs implied by D. If Q is D-satisfiable, then $chase_C(Q)$ is isomorphic to a subtree of a tree in $SAT(D)$.*

Example 9. Once again, the above result does not hold for DTDs which are not duplicate-free. Consider the DTD D from Example 2. Although in D each a can have either one or two b children, if an a has d child, then it has at most one b child. For queries $P = a[d][b[e][g]]$ and $Q = a[d][b/e][b/g]$, such a constraint implies that $P \supseteq_{SAT(D)} Q$. There is no containment mapping from P to $Q = chase_C(Q)$, the problem being that $chase_C(Q)$ is not isomorphic to a subtree of a tree in $SAT(D)$. □

Let C be the set of SCs and FCs implied by DTD D. Let Q be D-satisfiable, and $R \subseteq SAT(D)$ be the set of trees with a subtree isomorphic to $chase_C(Q)$. We call R the *C-satisfying set* for Q. Each tree in R has a *core* subtree which is isomorphic to $chase_C(Q)$, and each node in the core subtree is called a *core* node. Each node which is not a core node is called a *non-core* node.

Lemma 7. *Let D be a duplicate-free DTD and P and Q be queries in $XP^{\{[\,]\}}$. Let Q be D-satisfiable, C be the set of SCs and FCs implied by D, and R be the C-satisfying set for Q. If $P \supseteq_{SAT(D)} Q$, then, for each node w in P, either w can be mapped to a core node in every tree in R or w can be mapped to a non-core node in every tree in R.*

Lemma 8. *For $XP^{\{[\,]\}}$ queries P and Q, $P \supseteq_{SAT(C)} Q$ if and only if $P \supseteq chase_C(Q)$.*

Theorem 2. *Let D be a duplicate-free DTD and C be the set of SCs and FCs implied by D. For $XP^{\{[\,]\}}$ queries P and Q, $P \supseteq_{SAT(D)} Q$ if and only if $P \supseteq_{SAT(C)} Q$.*

Let D be a duplicate-free DTD and C be the set of SCs and FCs implied by D. The number of constraints in C can be exponential in the size of D, and $chase_C(Q)$ can have a number of nodes which is exponential in the number of constraints in C. However, in order to test $P \supseteq_{SAT(D)} Q$ for $\mathrm{XP}^{\{[\,]\}}$ queries P and Q, we do not have to generate all of C from D, nor do we have to chase Q with every constraint in C.

Instead we can first chase Q with each of the FCs in C, a step which can clearly be done in PTIME. Since applying an SC to Q never introduces a child with the same label as one of its siblings, there is no need to apply FCs after applying SCs. Now simultaneous top-down traversals of P and Q can determine nodes x in P and u in Q such that $\lambda(x) = \lambda(u)$, x has child y and u has no child with label $\lambda(y)$. Let B denote the set of labels of children of u in Q. We can then determine if D implies the SC $\lambda(u) : B \Downarrow \lambda(y)$ in PTIME, since D is duplicate-free [16]. If so, then we add a node v as a child of u with $\lambda(v) = \lambda(y)$. In this way, if P has n nodes, we can never add more than n^2 nodes to Q.

5 Limitations of Constraints

In this section, we address the question of whether there are simple constraints other than sibling and functional constraints that are necessary and sufficient for deciding D-containment in PTIME when neither the DTD nor the queries in $\mathrm{XP}^{\{[\,]\}}$ are duplicate-free.

Given a regular expression R, let $w \in L(R)$. Since queries in $\mathrm{XP}^{\{[\,]\}}$ cannot query the order of symbols in w, we adopt a bag representation to indicate the insignificance of ordering in the symbols of w: the notation $[w]$ denotes the bag of symbols appearing in w. If symbol a_i appears m_i times in w, $1 \leq i \leq k$, then

$[w] = \{a_1^{m_1}, \ldots, a_k^{m_k}\}$. The *unordered regular language* denoted by R, written $UL(R)$, is defined as $UL(R) = \{[w] \mid w \in L(R)\}$.

Let Σ be the set of symbols used in R, and w be an arbitrary string over Σ. We now show how to construct a DTD D that uses R, along with queries Q_1 and Q_2 in $\mathrm{XP}^{\{[\,]\}}$, such that $Q_1 \supseteq_{SAT(D)} Q_2$ if and only if $[w] \notin UL(R)$. Of course none of D, Q_1 or Q_2 is duplicate-free. This effectively demonstrates that using constraints less powerful than those which characterize unordered regular languages cannot provide a necessary and sufficient condition for query containment for $\mathrm{XP}^{\{[\,]\}}$.

Theorem 3. *Let R be a regular expression, Σ be the set of symbols used in R, and w be a string over Σ. There is a DTD D with content model R for some element, along with queries Q_1 and Q_2 in $XP^{\{[\,]\}}$, such that $Q_1 \supseteq_{SAT(D)} Q_2$ if and only if $[w] \notin UL(R)$.*

6 Related Work

The most recent related work is in the papers [1,7,9,10,16]. For containment of queries in the absence of constraints, containment was shown to be in PTIME for $\mathrm{XP}^{\{[\,],//\}}$ in [1], while it was shown to be CONP-complete for $\mathrm{XP}^{\{[\,],*,//\}}$ in [9].

Containment under constraints was shown to be undecidable in [7], when the XPath fragment includes $\mathrm{XP}^{\{[\,],*,//\}}$ along with disjunction, variable binding and equality testing, and the constraints include so-called bounded simple XPath integrity constraints (SXICs) and those (unbounded) constraints implied by DTDs. On the other hand, containment under DTDs was shown to be CONP-complete for $\mathrm{XP}^{\{[\,]\}}$ in [10,16]. In [1], child constraints along with *descendant* and *type co-occurrence* constraints, are used to minimise queries in $\mathrm{XP}^{\{[\,],//\}}$ in PTIME. A comprehensive classification of the complexity of containment of XPath fragments under DTDs is given in [10].

Earlier relevant work includes that by Calvanese *et al.* who proved that containment of conjunctions of regular path queries, a language more powerful than $\mathrm{XP}^{\{[\,],*,//\}}$, is decidable [5]. Papakonstantinou and Vassalos studied rewriting a query expressed in a language called TSL in terms of a set of views [11]. They include some rewritings based on DTD constraints. Finally, Böhm *et al.* derived constraints from DTDs in order to optimise expressions in the PAT algebra [3].

7 Conclusion

In this paper, we have studied the problem of query containment under DTDs for various fragments of XPath. By showing that containment under DTDs is decidable for $\mathrm{XP}^{\{[\,],*,//\}}$, we have contributed to solving the open problem stated in [7]. We have also identified that containment under duplicate-free DTDs for queries in $\mathrm{XP}^{\{[\,]\}}$ is tractable, while for queries and DTDs which are not duplicate-free, it remains intractable.

Further work is obviously needed to solve the issue of decidability of containment under DTDs for larger XPath fragments.

References

1. S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Minimization of tree pattern queries. In *Proc. ACM SIGMOD Conf.*, pages 497–508, 2001.
2. J. Bailey, A. Poulovassilis, and P. T. Wood. An event-condition-action language for XML. In *Proc. 11th Int. Conf. on the World Wide Web*, pages 486–495, 2002.
3. K. Böhm, K. Aberer, M. T. Özsu, and K. Gayer. Query optimization for structured documents based on knowledge of the document type definition. In *Proc. IEEE Advances in Digital Libraries*, pages 196–205, 1998.
4. A. Bruggemann-Klein, M. Murata, and D. Wood. Regular tree languages over non-ranked alphabets. Available at `ftp://ftp11.informatik.tu-muenchen.de/pub/misc/caterpillars/`, 1998.
5. D. Calvanese, G. de Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. 17th ACM Symp. on Principles of Databases Systems*, pages 149–158, 1998.
6. B. Chidlovskii. Using regular tree automata as XML schemas. In *Proc. IEEE Advances in Digital Libraries*, pages 89–104, 2000.
7. A. Deutsch and V. Tannen. Containment and integrity constraints for XPath fragments. In *Proc. 8th Int. Workshop on Knowledge Representation Meets Databases*, 2001.
8. D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. on Database Syst.*, 4(4):455–469, 1979.
9. G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proc. 21st ACM Symp. on Principles of Databases Systems*, 2002.
10. F. Neven and T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *Proc. 9th Int. Conf. on Database Theory*, 2003.
11. Y. Papakonstantinou and V. Vassalos. Query rewriting for semistructured data. In *Proc. ACM SIGMOD Conf.*, pages 455–466, 1999.
12. Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proc. 19th ACM Symp. on Principles of Databases Systems*, pages 35–46, 2000.
13. H. Seidl. Deciding equivalence of finite tree automata. *SIAM J. Computing*, 19(3):424–437, June 1990.
14. P. Wadler. A formal semantics of patterns in XSLT. In *Markup Technologies*, pages 183–202, 1999.
15. P. T. Wood. On the equivalence of XML patterns. In *Proc. 1st Int. Conf. on Computational Logic*, LNAI 1861, pages 1152–1166, 2000. Springer-Verlag.
16. P. T. Wood. Minimising simple XPath expressions. In *Proc. WebDB 2001: Fourth Int. Workshop on the Web and Databases*, pages 13–18, 2001.
17. World Wide Web Consortium. XML Path Language (XPath), Version 1.0. See `http://www.w3.org/TR/xpath`, November 1999. W3C Recommendation.
18. World Wide Web Consortium. XSL Transformations (XSLT), Version 1.0. See `http://www.w3.org/TR/xslt`, November 1999. W3C Recommendation.
19. World Wide Web Consortium. XQuery 1.0: An XML Query Language. See `http://www.w3.org/TR/xquery`, June 2001. W3C Working Draft.
20. M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. 7th Int. Conf. on Very Large Data Bases*, pages 82–94, 1981.