A Rewrite Approach for Pattern Containment  
— Application to Query Evaluation on  
Compressed Documents

Barbara Fila–Kordy  
barbara.kordy@univ-orleans.fr  
LIFO - Université d’Orléans, France

Abstract. In this paper we introduce an approach that allows to han-
dle the containment problem for the fragment XP(/, //, [ ], *) of XPath. 
Using rewriting techniques we define a necessary and sufficient condi-
tion for pattern containment. This rewrite view is then adapted to query 
evaluation on XML documents, and remains valid even if the documents 
are given in a compressed form, as dags.

1 Introduction

The focus in this paper is on the containment problem ([11, 12]) for the fragment 
XP(/, //, [ ], *) of XPath. XPath ([17]) is the main language for navigating and 
selecting nodes in XML documents. The segment XP(/, //, [ ], *) defines a class 
of Core XPath queries expressing descendant relationships between nodes, pos-
sibly containing filters, and allowing to use the don’t–care (or wildcard) symbol ‘*’. The queries of this fragment can be modeled by patterns: tree like graphs 
having two types of edges child and descendant. Every XML document \( t \) is an 
unranked tree \( t = (\text{Nodes}_t, \text{Edges}_t) \), and can also be seen as a pattern. For any 
two patterns \( P \) and \( Q \), we say that \( P \) is contained in \( Q \) \( (P \subseteq Q) \), if the query 
represented by \( Q \) is more general that the one represented by \( P \). For example, 
\( a/b \) is contained in \( a//b \), since \text{child} (/) is a particular case of \text{descendant} (/).

The big interest in the query containment problem ([11, 12, 16, 15]) is moti-
vated by its applications. Using the notion of pattern containment we can define 
queries which are equivalent, i.e., that on any XML document, select the same 
set of nodes. The query equivalence problem is closely linked to the query mini-
mization problem, which is essential for data base researchers. Since the time 
required for the evaluation of a given query \( Q \) is linear with respect to the size 
of \( Q \) ([8]), the minimization — possibility of replacing \( Q \) by an equivalent query 
of smaller size — is of interest from the point of view of complexity ([6, 1, 9, 14]).

We propose to handle the containment problem using a rewrite approach. 
We define a set of rewrite rules based on the semantics of XP(/, //, [ ], *)–query 
containment, and show that for any two patterns \( P \) and \( Q \), \( P \) is contained 
in \( Q \) \text{if and only if} we can rewrite \( P \) to \( Q \) using these rules. Such a rewrite 
view gives us a uniform framework to treat also other problems, for instance
query evaluation. We extend our approach on compressed documents encoded as straightline regular grammars, and apply our rewrite technique in order to evaluate $\text{XP}(/, //,[ ],* )$–queries on compressed or unfolded (arborescent) XML documents.

This paper is organized as follows: In Section 2 we introduce terminology and notation, and recall some results on the pattern containment problem. Our rewrite method is presented in Section 3. Finally, in Section 4 we show how to adapt this rewrite approach to the query evaluation problem.

2 The Pattern Containment Problem

Let $\Sigma$ be an alphabet containing the element names of all XML documents considered. In this work we consider the fragment $\text{XP}(//,[ ],* )$ of XPath, which consists of: node tests (symbols from $\Sigma \cup \{\ast\}$), child axis (/), descendant axis (//), and qualifiers also called filters ([...]). Any element of $\text{XP}(//,[ ],* )$ is a query that can be represented as a rooted tree structure graph over $\Sigma \cup \{\ast\}$, called unary pattern, having:

- edges of two types: simple for child, and double for descendant,
- nodes labeled by the symbols from $\Sigma \cup \{\ast\}$,
- one distinguished node marked with a special selection symbol ‘$s$’ representing the output information.

For instance, the unary pattern in Figure 1 represents the $\text{XP}(//,[ ],* )$ query $/a//b[./b/c/d]/c[./*/d]$. The notion of unary patterns is easily extended to

that of $n$–ary patterns, where we have $n$ distinguished nodes, that model $n$–ary queries selecting $n$–tuples of nodes. Miklau and Suciu show in [11] that, for the purpose of the containment problem, it is sufficient to consider only the patterns of arity zero, called boolean, where there are no distinguished nodes. Thus, all patterns considered in the sequel will be boolean, and they will be simply called patterns.

For a given pattern $P$, we denote by $\text{Nodes}_P$ the set of all its nodes. For any $u \in \text{Nodes}_P$, $\text{name}_P(u)$ stands for the element of $\Sigma \cup \{\ast\}$ labeling the node $u$. By

![Fig. 1. Unary pattern representing query /a//b[./b/c/d]/c[./*/d]](image)
Edges \( (P) \) and Edges \( \Downarrow (P) \) we mean respectively the set of child and descendant edges of \( P \). We define the size of \( P \) (denoted by \( |P| \)) to be the number of all edges in \( P \).

**Definition 1.** An XML tree \( t \) is a model of a pattern \( P \) iff there exists an embedding from \( P \) to \( t \); i.e., a function \( e: \text{Nodes}_P \rightarrow \text{Nodes}_t \), satisfying the following conditions:

1. \( e \) preserves the root: \( e(\text{root}_P) = \text{root}_t \);
2. \( e \) preserves the names:
   \[ \forall u \in \text{Nodes}_P, \text{name}_P(u) = *, \text{or} \text{name}_P(u) = \text{name}_t(e(u)) \];
3. \( e \) preserves the relation child:
   \[ \forall (u, v) \in \text{Edges}_1(P), (e(u), e(v)) \in \text{Edges}_t \];
4. \( e \) preserves the relation descendant:
   \[ \forall (u, v) \in \text{Edges}_y(P), (e(u), e(v)) \in (\text{Edges}_t)^+ \],

where \( (\text{Edges}_t)^+ \) is the transitive closure of the relation \( \text{Edges}_t \).

The notion of model is illustrated in Figure 2.

![Fig. 2. Pattern \( P \), its model \( t \), and embedding \( e \) from \( P \) to \( t \)](image)

**Definition 2.** Given two patterns \( P \) and \( Q \), we say that \( P \) is contained in \( Q \) \( (P \subseteq Q) \) iff every model of \( P \) is also a model of \( Q \). The patterns \( P \) and \( Q \) are equivalent \( (P \equiv Q) \) iff \( P \subseteq Q \) and \( Q \subseteq P \).

Figure 3 represents two patterns which are easily seen to be equivalent.

![Fig. 3. Equivalent patterns \( P \) and \( Q \)](image)

Miklau and Suciu prove in [11] that the containment problem for XP(\( /,\//,[\[\].*,\]) is CoNP-complete. They also give a sufficient — but not necessary —
condition for pattern containment. For that purpose, they extend the notion of embedding to pattern homomorphism:

**Definition 3.** Given two patterns $P$ and $Q$, a homomorphism from $Q$ to $P$ is a function $\varphi : \text{Nodes}_Q \rightarrow \text{Nodes}_P$, which is:

- root and name preserving;
- child preserving:
  $$\forall (u, v) \in \text{Edges}_1(Q), (\varphi(u), \varphi(v)) \in \text{Edges}_1(P);$$
- descendant preserving:
  $$\forall (u, v) \in \text{Edges}_b(Q), (\varphi(u), \varphi(v)) \in (\text{Edges}_1(P) \cup \text{Edges}_b(P))^+. $$

The authors of [11] prove that if there exists a homomorphism from $Q$ to $P$, then $P$ is contained in $Q$. They give an algorithm which for two given patterns $P$ and $Q$ verifies, in time $O(|P||Q|)$, whether there exists a homomorphism from $Q$ to $P$. Figure 4 shows the patterns $P$ and $Q$, and the homomorphism $\varphi$ from $Q$ to $P$ proving that $P \subseteq Q$. Nevertheless, the existence of a homomorphism from $Q$ to $P$ is not a necessary condition for $P \subseteq Q$ (as is easily checked for the patterns $P$ and $Q$ given in Figure 3, which are equivalent, but there is no homomorphism neither from $Q$ to $P$, nor from $P$ to $Q$). In the following example we show a way to prove the containment $P \subseteq Q$, if there is no homomorphism from $Q$ to $P$.

**Example 1.** In Figure 5 we have presented two patterns $P$ and $Q$ (borrowed from [11]) satisfying $P \subseteq Q$, such that there is no homomorphism from $Q$ to $P$.

![Figure 4](image1.png)

**Fig. 4.** Homomorphism $\varphi$ from $Q$ to $P$ proving that $P \subseteq Q$

![Figure 5](image2.png)

**Fig. 5.** Patterns s.t. $P \subseteq Q$, but no homomorphism from $Q$ to $P$

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

Here, to show the containment $P \subseteq Q$, we have to reason by cases. Let $t$ be a model
of $P$, and consider the middle edge $c/d$ of pattern $P$. This edge can be realized on $t$:

- either by the child edge $c/d$ (as in Figure 6),

![Fig. 6. Model of $P$ (and $Q$), where $c/d$ is realized by $c/d$](image)

- or by a path $c/*/\ldots/d$, having length $\geq 2$ (as in Figure 7).

![Fig. 7. Model of $P$ (and $Q$), where $c/d$ is realized by $c/a/d$](image)

Such an analysis shows that any model of $P$ is also a model of $Q$, thus $P \subseteq Q$. However, it is impossible to define one general homomorphism from $Q$ to $P$, as the right branch $a//b/c/*/d$ of $Q$, corresponds in each case to a different branch of $P$.

3 Pattern Containment via rewriting

We propose to handle the pattern containment problem using an approach based on rewriting techniques. A key idea is that checking containment requires case analysis in general, and this can be encoded as rewriting (as we illustrate in Example 2 below). We construct a rewrite system $\mathcal{R}$ that permits to define a necessary and sufficient condition (see Theorem 1) for pattern containment on the fragment $\text{XP}(\,,//,\,[\,],\,*)$. 
We start by giving a formal definition of pattern, alternative to that used in the previous sections.

**Definition 4.** We define patterns over an alphabet \( \Sigma \) as the expressions \( P \) derived from the grammar of Table 1, where \( \downarrow \) and \( \Downarrow \) stand respectively for child and descendant. \( \omega \in \Sigma \cup \{\ast\} \), and ‘\( \ast \)’ is the don’t-care symbol of XPath that can replace any \( \sigma \) of \( \Sigma \).

\[
\begin{align*}
M : & \quad \varepsilon \mid \downarrow \omega \mid \Downarrow \omega \mid MM \quad \text{// path} \\
S : & \quad \emptyset \mid \{MS\} \mid S \cup S \quad \text{// set of sibling unrooted terms} \\
P : & \quad \omega MS \quad \text{// patterns}
\end{align*}
\]

Table 1. Grammar for patterns

This grammar produces precisely the patterns as defined in [6, 11]. For instance, the graph \( P \) in Figure 8 corresponds to the expression

\[
P = a \Downarrow b \{ \downarrow b \downarrow c \downarrow d, \downarrow c \Downarrow d \}, \downarrow c \Downarrow \ast \Downarrow d
\]

derived from the grammar of Table 1.

![Fig. 8. Pattern](image)

By a *term* we mean any expression of the type \( M \), \( S \) or \( P \) derived from the grammar of Table 1, as well as any finite disjunction \( P_1 \lor P_2 \lor \cdots \lor P_n \) of patterns. The terms of the type \( M \) correspond to the linear paths without branching, they start by a modal symbol \( \xi \in \{\downarrow, \Downarrow\} \); those of the type \( S \) represent a set of terms having a common parent node; and those of the type \( P \) are patterns. The terms in \( P \) are *rooted* (they start by a symbol from \( \Sigma \cup \{\ast\} \)), those in \( M \) and \( S \) are *unrooted*. To simplify, we will often identify the singleton \( \{MS\} \) with the term \( MS \). Given patterns \( P \) and \( P_i \), for \( 1 \leq i \leq n \), the terms of the form \( \varepsilon, P \), or \( P_1 \lor \cdots \lor P_n \), will also be called *d-patterns*. A tree \( t \) is a model of a d-pattern \( P_1 \lor \cdots \lor P_n \) iff \( t \) is a model of at least one pattern \( P_i \), for \( 1 \leq i \leq n \). A disjunctive
d–pattern will be used in case analysis to represent different models of a given pattern with a unique term, as in the following example.

**Example 2.** Consider the patterns

\[ P = f \downarrow * \downarrow a \quad \text{and} \quad Q = f \downarrow * \downarrow a \]

given in Figure 3. We know that \( P \equiv Q \), thus in particular \( P \subseteq Q \), but there is no homomorphism which proves it. Using the rules of our system \( \mathcal{R} \) defined below we will be able to rewrite \( P \) to \( Q \), and prove the containment \( P \subseteq Q \). The idea is that every descendant is either a child or has a depth \( \geq 2 \); thus, the edge \(* \downarrow a\) of \( P \) can be realized either by the child edge \( * \downarrow a\), or by a path having at least one additional node between ‘\*' and ‘\a’, that we can denote by \(* \downarrow * \downarrow a\).

We will then rewrite the pattern \( P \) to the d–pattern

\[ f \downarrow * \downarrow a \lor f \downarrow * \downarrow a \]

depicting the two cases mentioned. The two pattern components of this d–pattern will then be rewritten in parallel. A child, as well as a descendant of depth \( \geq 2 \) are particular cases of descendant. As a consequence, the edge \( f \downarrow * \) will be rewritten to \( f \downarrow * \), idem for the path \( f \downarrow * \downarrow \). This will give us the following term:

\[ f \downarrow * \downarrow a \lor f \downarrow * \downarrow a, \]

which will be finally rewritten to \( Q \), since each pattern composing this d–pattern is exactly the pattern \( Q \).

To formalize the idea employed in the examples above, we introduce a set \( \mathcal{R} \) of rules that serve to rewrite rooted and unrooted terms. Let \( M, S \) (possibly with primes, subscripts) be as in the grammar of Table 1; \( \xi, \xi' \in \{\downarrow, \downarrow\} \), \( \sigma \in \Sigma \), and \( \omega, \omega' \in \Sigma \cup \{\ast\} \):

1. \( S \rightarrow \emptyset \), \( M \rightarrow \varepsilon \) //cut;
2. \( M \sigma S \rightarrow M \ast S \) //replace any symbol of \( \Sigma \) by the ‘\*’ of XPath;
3. \( \uparrow \omega S \rightarrow \uparrow \omega S \) //every child is also a descendant;
4. \( \xi \omega \xi' \omega' S \rightarrow \uparrow \omega' S \) //ignore an intermediate node;
5. \( M\{S_1, S_2\} \rightarrow \{MS_1, MS_2\} \) //left distributivity;
6. \( S \rightarrow S' \cup S \), where \( S \rightarrow S' \) //add new siblings;
7. \( S \cup S_1 \rightarrow S' \cup S_1 \), if \( S \rightarrow S' \) //rewrite some of the siblings;
8. \( \downarrow \omega S \rightarrow (\downarrow \omega S) \lor (\downarrow \ast \downarrow \omega S) \) //case analysis: descendant is either a child or has depth \( \geq 2 \);
9. \( \downarrow \omega S \rightarrow (\downarrow \omega S) \lor (\downarrow \ast \downarrow \omega S) \) //idem.

By **context–pattern** we mean any pattern having a special additional hole symbol \( \Diamond \) that replaces one of its unrooted sub–terms. Let us consider a context–pattern \( \mathcal{C} \) and an unrooted term \( X \). We define the fill–in of \( \mathcal{C} \) with \( X \) (denoted as \( \mathcal{C} \uparrow X \)) to be the pattern obtained from \( \mathcal{C} \) by replacing its hole symbol with the term \( X \); e.g. for the context–pattern \( \mathcal{C} = f\{\downarrow a, \downarrow b(\downarrow, \downarrow d), \downarrow \ast\} \), and the unrooted term \( X = \downarrow x\{\downarrow y, \downarrow z\} \), we get the fill–in:

\[ \mathcal{C} \uparrow X = f\{\downarrow a, \downarrow b\{\downarrow x\{\downarrow y, \downarrow z\}, \downarrow d\}, \downarrow \ast\}. \]
We also suppose that for any context–pattern $C$ and unrooted terms $X$ and $X'$, the notation $\diamondsuit C (X \lor X')$ stands for the disjunctive d–pattern $C \diamondsuit X \lor C \diamondsuit X'$. To rewrite patterns with the rules of $\mathcal{R}$ we use suffix rewriting:

**Definition 5.** Given a pattern $P$ and a pattern or a d–pattern $Q$, we say that $P$ can be rewritten to $Q$ in one step using suffix rewriting (denoted as $P \longrightarrow^* \mathcal{R} Q$), if there exist a context–pattern $C$ and two unrooted terms $X$ and $X'$, such that: $P = C \diamondsuit X$, $Q = C \diamondsuit X'$, and $X \longrightarrow X'$ is an instance of a rule in $\mathcal{R}$.

Moreover, disjunctive terms can be rewritten using the following additional two rules, where $P$ is a pattern, and $D, D_1, D_2$ stand for d–patterns:

10. $D_1 \lor D \longrightarrow D_2 \lor D$ if $D_1 \longrightarrow D_2$ //case rewriting;
11. $P \lor P \lor D \longrightarrow P \lor D$ //consider any given case only once.

The main result of our work is the following:

**Theorem 1.** For any two patterns $P$ and $Q$, $P$ is contained in $Q$ if and only if $P \longrightarrow^* \mathcal{R} Q$, i.e., $P$ can be rewritten to $Q$ using the rules of $\mathcal{R}$ in zero or finitely many steps.

**Proof.** The semantics of the rules in $\mathcal{R}$ guarantee that $P \longrightarrow^* \mathcal{R} Q$ implies $P \subseteq Q$. To show the converse, we start with the following lemma:

**Lemma 1.** For any patterns $P$ and $Q$, if there exists a homomorphism from $Q$ to $P$, then $P \longrightarrow^* \mathcal{R} Q$.

**Proof.** Given a homomorphism $\varphi$ from $Q$ to $P$, we construct a pattern $P'$, such that $P \longrightarrow^* \mathcal{R} P'$, as follows:

(a) for every node $u$ of $Q$, we construct a corresponding node $u'$ of $P'$, and we set $\text{name}_{\mathcal{P}}(u') = \text{name}_{\mathcal{P}}(\varphi(u))$;
(b) we construct a child edge $(u', v') \in \text{Edges}_1(P')$, if and only if $(u, v) \in \text{Edges}_1(Q)$;
(c) we construct a descendant edge $(u', v') \in \text{Edges}_2(P')$, if and only if $(u, v) \in \text{Edges}_2(Q)$.

The cost of such a construction is linear with respect to the size of $Q$. The pattern $P'$ can be rewritten to the pattern $Q$ using rule 2 of $\mathcal{R}$. Indeed, the structures (nodes, simple and double edges) of $P'$ and $Q$ are the same, but the names of some $u \in \text{Nodes}_Q$ and the corresponding node $u' \in \text{Nodes}_{P'}$ may be different. Condition (a) implies that: either $\text{name}_Q(u) = \text{name}_{\mathcal{P}}(u') = \text{name}_{\mathcal{P}}(\varphi(u))$, or $\text{name}_Q(u) \neq \text{name}_{\mathcal{P}}(u') = \text{name}_{\mathcal{P}}(\varphi(u))$. In the second case we have (see Definition 3): $\text{name}_Q(u) = ^*$, and $\text{name}_{\mathcal{P}}(u') \in \Sigma$, thus to rewrite $P'$ to $Q$ we have to use rule 2.

It remains to be shown that $P$ can be rewritten to $P'$:

– using rules 1 and 7 (with $S' = \emptyset$), we can ignore all sub–branches of $P$ which do not contain the nodes images under $\varphi$;
if some node \( w \) of \( P \) is an image of \( m \) distinct nodes \( u_1, \ldots, u_m \) of \( Q \), then we rewrite the unique node \( w \) of \( P \) to \( m \) nodes \( u'_1, \ldots, u'_m \) of \( P' \), by using rule 6 (with \( S' = S \)) and/or rule 5.

- case when edge \((u', v')\) is in \( \text{Edges}_{\delta}(P') \): from condition (b) we know that \((u, v) \in \text{Edges}_{\delta}(Q)\), thus by Definition 3 we have \((\varphi(u), \varphi(v)) \in \text{Edges}_{\delta}(P)\) (we have nothing to do with the edge \((\varphi(u), \varphi(v))\) when rewriting \( P \) to \( P' \));

- case when edge \((u', v')\) is in \( \text{Edges}_{\delta}(P') \): from condition (c) and Definition 3 we can deduce that there exist \( k \geq 1 \) and \( w_0, \ldots, w_k \in \text{Nodes}_P \), such that: \( w_0 = \varphi(u), w_k = \varphi(v) \), and \( \forall i \in \{0, \ldots, k-1\} \) we have \((w_i, w_{i+1}) \in \text{Edges}_{\delta}(P) \cup \text{Edges}_{\delta}(P)\). If \( k = 1 \) and \((\varphi(u), \varphi(v)) \in \text{Edges}_{\delta}(P)\), then we can rewrite \( P \) to \( P' \) using rule 3. If \( k \geq 2 \), then we use \((k - 1 \text{ times})\) rule 4 to ignore the nodes \( w_1, \ldots, w_{k-1} \) while rewriting \( P \) to \( P' \).

Finally, we obtain \( P \xrightarrow{*} R P' \xrightarrow{*} R Q \). \( \square \)

Note that if \( P \) is a tree, we also have the converse of Lemma 1; of course, in this case a homomorphism from \( Q \) to \( P \) is an embedding from the pattern \( Q \) to the tree \( P \). Thus we have the following:

**Remark 1.** A tree \( t \) is a model of a pattern \( Q \) iff \( t \xrightarrow{*} R Q \).

By a *homomorphism* from a pattern \( Q \) to a d–pattern \( D = P_1 \lor \cdots \lor P_n \), we mean a function which is a homomorphism from \( Q \) to \( P_i \), for every \( 1 \leq i \leq n \). Thus, using Lemma 1, we obtain the following corollary:

**Corollary 1.** For any given pattern \( Q \) and a d–pattern \( D \), if there exists a homomorphism from \( Q \) to \( P_i \), for every \( 1 \leq i \leq n \), then \( D \xrightarrow{*} R Q \).

**Proof.** It suffices to remark that rules 10 and 11 imply that a d–pattern \( P_1 \lor \cdots \lor P_n \) can be rewritten to a pattern \( Q \) if and only if, for every \( 1 \leq i \leq n \), we have \( P_i \xrightarrow{*} R Q \). \( \square \)

To finish the proof of Theorem 1, we use the following proposition:

**Proposition 1.** For two patterns \( P \) and \( Q \), if \( P \subseteq Q \), then one can construct a d–pattern \( D \) verifying \( P \xrightarrow{*} R D \), such that there exists a homomorphism from \( Q \) to \( D \).

**Proof.** From the result of Miklau and Suciu ([11]) we know that it is possible to check if there exists a homomorphism from \( Q \) to \( P \). If it is the case, the d–pattern \( D \) satisfying the proposition is equal to \( P \) (see Lemma 1). If not, a disjunctive d–pattern \( D = P_1 \lor \cdots \lor P_n \) representing case analysis, such that for every \( 1 \leq i \leq n \), there exists a homomorphism from \( Q \) to \( P_i \).

This terminates the proof of Theorem 1.
The rewrite system $\mathcal{R}$ is non-deterministic; nevertheless if $P$ and $Q$ are given, there exists a well-defined, goal-directed strategy for rewriting $P$ to $Q$. The idea is to use only those rules among 1–11 that permit to converge to $Q$. We illustrate this strategy in the following example:

**Example 3.** Let $P$ and $Q$ be the patterns represented in Figure 5. We show how to rewrite $P$ to $Q$, and thus prove the containment $P \subseteq Q$. The pattern $P = a \downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \downarrow d\}, \downarrow c \downarrow * \downarrow d\}$ can be seen as the fill-in

$$a \downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \downarrow d\}, \downarrow c \downarrow * \downarrow d\} \downarrow d.$$

Using rule 8 for the underlined term, we encode the cases depicted in Example 1:

$$a \downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \downarrow d\}, \downarrow c \downarrow * \downarrow d\} \downarrow d$$
$$\lor$$
$$a \downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \downarrow d\}, \downarrow c \downarrow * \downarrow d\} \downarrow * \downarrow d.$$

We obtain the d–pattern

$$a \downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \downarrow d\}, \downarrow c \downarrow * \downarrow d\} \downarrow b \downarrow c \downarrow d$$
$$\lor$$
$$a \downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \downarrow d\}, \downarrow c \downarrow * \downarrow d\}, \downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \downarrow d\} \downarrow c \downarrow * \downarrow d.$$

which can be seen under the form

$$a \downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \downarrow d\}, \downarrow c \downarrow * \downarrow d\} \downarrow b \downarrow c \downarrow d$$
$$\lor$$
$$a \downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \downarrow d\}, \downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \downarrow d\}, \downarrow c \downarrow * \downarrow d\} \downarrow c \downarrow * \downarrow d.$$

We rewrite it using rule 10. We cut (rule 1) the underlined parts, and get

$$a \downarrow b\{\downarrow b \downarrow c \downarrow d\}, \downarrow c \downarrow * \downarrow d\} \lor a \downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \downarrow d\} \downarrow c \downarrow * \downarrow d\}.$$

The d–pattern that we have obtained is then identified with

$$a \downarrow b\{\downarrow b \downarrow c \downarrow d\}, \downarrow c \downarrow * \downarrow d\} \lor a \downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \downarrow d\} \downarrow c \downarrow * \downarrow d\}.$$

Its first component is equal to the pattern $Q$. To the second one, seen as the fill-in $a \Downarrow \downarrow b\{\downarrow b \downarrow c \downarrow d\}, \downarrow c \downarrow * \downarrow d\}$, we apply rule 4, and get the term $a \Downarrow \downarrow b\{\downarrow b \downarrow c \downarrow d\}, \downarrow c \downarrow * \downarrow d\} = a \downarrow b\{\downarrow b \downarrow c \downarrow d\}, \downarrow c \downarrow * \downarrow d\}$. Thus we obtain the d–pattern

$$a \downarrow b\{\downarrow b \downarrow c \downarrow d\}, \downarrow c \downarrow * \downarrow d\} \lor a \downarrow b\{\downarrow b \downarrow c \downarrow d\}, \downarrow c \downarrow * \downarrow d\} = Q \lor Q,$$

that is finally rewritten to $Q$ using rule 11.

**Remark 2.** Our approach is no longer valid, if it is not based on suffix rewriting; e.g. for $P = \ast \downarrow \ast$ and $Q = \ast \downarrow \ast$, we have $P \subseteq Q$ ($P \xrightarrow{\mathcal{R}} Q$ using rules 9, 1), but $P\Downarrow \downarrow a = \ast \downarrow \ast \downarrow a$ is not contained in $Q\Downarrow \downarrow a = \ast \downarrow \ast \downarrow a$: for instance, the tree $t = f \downarrow g \downarrow b \downarrow a$ is a model of $\ast \downarrow \ast \downarrow a$, but not of $\ast \downarrow \ast \downarrow a$. 

10
4 Applications

The objective of this section is to show that our rewrite approach remains valid even if the models of patterns are given in a compressed form (as dags), and that it can be adapted for query evaluation on XML documents.

4.1 Case of Compressed Documents

To model compressed documents we use rooted dags instead of trees (as in [5, 2, 10, 7]). Figure 9 represents three formats of the same document: tree, fully and partially compressed format (see [5] for formal definitions). In the sequel, by $t$ we will denote any given representation (tree or dag) of the document considered. To distinguish between different formats of the same document we use regular tree grammars. Given a document $t$, we call normalized grammar for $t$ a regular tree grammar $G_t$:

– which recognizes only $t$,
– where every node of $t$ is represented by exactly one non–terminal,
– the indexes of non–terminals for children nodes are grater then the indexes of non–terminals for parent nodes.

Such normalized grammars are straightline in the sens defined in [3], i.e., there is no cycle on their dependency graph. For this reason we will refer to them as SLR grammars.

**Example 4.** The SLR grammars for the three dags from Figure 9 are respectively:

\[
\begin{align*}
X_0 & \to f(X_1, X_2, X_3, X_4) & Y_0 & \to f(Y_1, Y_1, Y_2, Y_1) & Z_0 & \to f(Z_1, Z_1, Z_2, Z_3) \\
X_1 & \to a & Y_1 & \to a & Z_1 & \to a \\
X_2 & \to a & Y_2 & \to b & Z_2 & \to b \\
X_3 & \to b & & & Z_3 & \to a. \\
X_4 & \to a \\
\end{align*}
\]

![Fig. 9. Tree, fully compressed format, partially compressed format](image-url)
We extend the notion of SLR grammar to patterns. To define a normalized grammar $G_P$ for pattern $P$, it is sufficient that every non–terminal $X_i$ appearing on the right hand side of any production of $G_P$, is preceded by a modal symbol ↓ or ⇓, corresponding to the type of edge pointing to the node represented by $X_i$ on $P$. In order to have a uniform notation that covers patterns as well as documents, we will do the same on the normalized grammar $G_t$, for any document $t$: every non–terminal $X_i$ appearing on the right hand side of some production in $G_t$, will be preceded by ↓. For instance, the grammars $G_P$ and $G_t$ respectively for the pattern $P$ and the tree $t$ of Figure 11, are given in Figure 10.

\[
P_0 \to \ast(\downarrow P_1, \downarrow P_2) \quad X_0 \to f(\downarrow X_2, \downarrow X_1)
\]
\[
P_1 \to a \quad X_1 \to b(\downarrow X_2)
\]
\[
P_2 \to a \quad X_2 \to a.
\]

Fig. 10. SLR grammars $G_P$ and $G_t$ for $P$ and $t$ from Figure 11

Remark that the notion of embedding from a pattern to a tree can be extended in a natural way to an embedding from a pattern to a rooted dag. To define an embedding $e$ from a pattern $P$ to a dag $t$, we replace the conditions 2 and 3 of Definition 1 respectively by:

3. $\forall u, v \in \text{Nodes}_P$, such that $(u, v) \in \text{Edges}_↓(P)$, there exists an edge going from $e(u)$ to $e(v)$ on $t$;
4. $\forall u, v \in \text{Nodes}_P$, such that $(u, v) \in \text{Edges}_⇓(P)$, all paths going from $e(u)$ to $e(v)$ on $t$ are in $(\text{Edges}_t)^+$.

Note that, if $t$ is a tree, such a definition is equivalent to Definition 1, since on any tree we have at most one path between two nodes. The notion of (dag) model of a pattern and the pattern containment problem are defined in the same way as in the case of tree models. Figure 11 shows a pattern $P$, its compressed model $t$, and an embedding from $P$ to $t$.

Fig. 11. Pattern $P$, its compressed model $t$, and embedding from $P$ to $t$
SLR grammars can be used in our rewrite approach. To prove that a given dag \( t \) is a model of a pattern \( P \), it is sufficient (according to Remark 1) to rewrite the grammar \( G_t \) representing \( t \) to the grammar \( G_P \) representing \( P \). We illustrate this idea in the following example.

**Example 5.** Consider the grammars \( G_P \) and \( G_t \) given in Figure 10. We show how to rewrite \( G_t \) to \( G_P \) using rules of \( \mathcal{R} \):

\[
\begin{align*}
X_0 &\rightarrow f(\downarrow X_2, \downarrow X_1) \quad 2\quad X_0 \rightarrow * (\downarrow X_2, \downarrow X_1) \quad 1\quad X_0 \rightarrow * (\downarrow X_2) \quad 6
\end{align*}
\]

\[
\begin{align*}
X_1 &\rightarrow b(\downarrow X_2) \quad 1\quad X_1 &\rightarrow b(\downarrow X_2) \quad 1
\end{align*}
\]

\[
X_2 \rightarrow a
\]

The first production of \( G_t \) is first rewritten using rule 2; then we cut a branch represented by \( X_1 \) (rule 1). At the same time, we can eliminate from \( G_t \) the production \( X_1 \rightarrow b(\downarrow X_2) \), since it has become unproductive (there is no more production having \( X_1 \) on their right hand sides).

\[
\begin{align*}
X_0 &\rightarrow * (\downarrow X_2, \downarrow X_2') \quad 3\quad X_0 &\rightarrow * (\downarrow X_2, \downarrow X_2') \approx P_0 \rightarrow *(\downarrow P_1, \downarrow P_2)
\end{align*}
\]

\[
\begin{align*}
X_2 &\rightarrow a \quad X_2' &\rightarrow a \approx P_1 \rightarrow a
\end{align*}
\]

Then, using rule 6 we double the number of children of \( X_0 \); we introduce a new non–terminal \( X_2' \), which produces the same sub–pattern as \( X_2 \). Finally, by Rule 3, we get a grammar which is equal, up to non–terminal renaming, to \( G_P \).

### 4.2 Query Evaluation

SLR grammars help us to adapt the rewrite approach of Section 3 to XP(//, [],*)–query evaluation on (compressed) documents. To represent unary queries, we use unary patterns (see Section 2). Let us consider the unary pattern \( P \) rep-

![Fig. 12. Unary pattern \( P \) and its compressed model \( t \)](image-url)
given in Figure 12. The corresponding SLR grammars $G_P$ and $G_t$ are respectively:

\[
\begin{align*}
P_0 &\rightarrow *() \quad & P_0 &\rightarrow f(\downarrow X_1, \downarrow X_2) \\
P_1 &\rightarrow c(\downarrow P_2, \downarrow P_3) \quad & X_0 &\rightarrow f(\downarrow X_1) \\
P_2(s) &\rightarrow d(\downarrow P_4) \quad & X_1 &\rightarrow a(\downarrow X_6, \downarrow X_3) \\
P_3 &\rightarrow c \quad & X_3 &\rightarrow c(\downarrow X_4, \downarrow X_5) \\
P_4 &\rightarrow a \quad & X_4 &\rightarrow d(\downarrow X_6) \\
X_5 &\rightarrow c(\downarrow X_6) \quad & X_5 &\rightarrow c(\downarrow X_6) \\
X_6 &\rightarrow a \\
\end{align*}
\]

The non–terminal $P_2$ of $G_P$ is marked ‘s’, since it represents the output node of $P$. To find an answer for $P$ on $t$, we rewrite the grammar $G_t$ to the grammar $G_P$, using the rules of $R$. The non–terminal of $G_t$ which will be rewritten to the selecting non–terminal $P_2$ of $G_P$, will represent an answer for $P$ on $t$. We illustrate this reasoning below:

\[
\begin{align*}
X_0 &\rightarrow f(\downarrow X_1, \downarrow X_2) \quad & X_0 &\rightarrow f(\downarrow X_1) \\
X_1 &\rightarrow a(\downarrow X_6, \downarrow X_3) \quad & X_1 &\rightarrow a(\downarrow X_3) \\
X_3 &\rightarrow c(\downarrow X_4, \downarrow X_5) \quad & X_3 &\rightarrow c(\downarrow X_4, \downarrow X_5) \\
X_4 &\rightarrow d(\downarrow X_6) \quad & X_4 &\rightarrow d(\downarrow X_6) \\
X_5 &\rightarrow c(\downarrow X_6) \quad & X_5 &\rightarrow c(\downarrow X_6) \\
X_6 &\rightarrow a \\
\end{align*}
\]

We have obtained an SLR grammar, which is (up to non–terminal renaming) the SLR grammar $G_P$ for $P$. The non–terminal $X_4$ of $G_t$ has been rewritten to the non–terminal $P_2$, thus the node represented by $X_4$ is an answer for $P$ on $t$.

Note that, as any query $P$ of the fragment $XP(/.,//,[@],*)$ is purely descendant, the answer for $P$ on a document $t$ does not depend on the form under which $t$ is given (tree or dag); this is no longer valid for queries containing descendant axes (cf.[5]). Remark also that our rewrite approach can be extended to any $n$–ary query of $XP(/.,//,[@],*)$; an $n$–ary query selects a set of $n$–tuples of nodes ([13]), and is easily represented as an $n$–ary pattern.
5 Conclusion

We have presented an approach based on rewrite techniques, that allows to handle the problem of query containment for the segment XP(/.,[ ],*\) of XPath. Such a rewrite view is also appropriate for compressed documents modeled as dags, and can be adapted to (unary as well as n–ary) query evaluation on (compressed) documents.

Straightline regular tree grammars can provide an exponential space compression. Nevertheless there exist more efficient compression techniques, like those based on straightline context–free grammars (SLCF, [4,3]), giving better (up to doubly exponential) compression rates. Currently we are studying the possibility of extending our rewrite approach to such more efficient compressions. We also hope to adapt our results to larger fragments of XPath, containing queries modeled by more general patterns, having both descendant and ascendant edges.

References


