A Rewrite Approach for Pattern Containment

Barbara Kordy
`barbara.kordy@univ-orleans.fr`

LIFO - Université d'Orléans, France

Abstract. In this paper we introduce an approach that allows to handle the containment problem for the fragment XP(`/`,`//`,`[ ]`,$*$) of XPath. Using rewriting techniques we define a necessary and sufficient condition 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 ([1,2]) for the fragment XP(`/`,`//`,`[ ]`,$*$) of XPath. XPath ([3]) 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, possibly 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 $t = (Nodes_t, Edges_t)$, and can also be seen as a pattern. For any two patterns P and Q, we say that P is contained in Q ($P \subseteq Q$), iff the query represented by Q is more general than the one represented by P. For example, a/b is contained in $a//b$, since `child` (`/`) is a particular case of `descendant` (`//`).

The big interest in the query containment problem ([1,2,4,5]) is motivated 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 minimization 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 ([6]), the minimization — possibility of replacing Q by an equivalent query of smaller size — is of interest from the point of view of complexity ([7,8,9,10]).

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 *if and only if* we can rewrite P to Q using these rules. This provides a characterization of the containment problem using algebraic techniques, which was missing in the literature. 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 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 Σ be an alphabet containing the element names of all XML documents considered. In this work we consider the fragment XP(/,//,[ ],$*$) of XPath, which consists of: node tests (symbols from $\Sigma \cup \{*\}$), `child` axis (/), `descendant` axis (//), and qualifiers also called filters ([...]). Any element of XP(/,//,[ ],$*$) is a query that can be represented as a rooted tree structure graph over $\Sigma \cup \{*\}$, called *unary pattern*, having:

- edges of two types: simple for *child*, and double for *descendant*,
- nodes labeled by the symbols from $\Sigma \cup \{*\}$,
- one distinguished node marked with a special selection symbol 's' representing the output information (located at the end of the main path in the query considered).

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

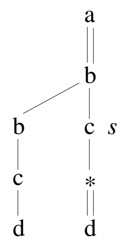

Fig. 1. Unary pattern representing query /a//b[./b/c/d]/c[./$*$//d]

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 [1] 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 $Nodes_P$ the set of all its nodes. For any $u \in Nodes_P$, $name_P(u)$ stands for the element of $\Sigma \cup \{*\}$ labeling the node u. By $Edges_{\downarrow}(P)$ 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 $\mathbf{e}\colon Nodes_P \to Nodes_t$, satisfying the following conditions:*

1. $\mathbf{e}$ *preserves the root:* $\mathbf{e}(root_P) = root_t$;
2. $\mathbf{e}$ *preserves the names:* $\forall u \in Nodes_P$, $name_P(u) = *$, *or* $name_P(u) = name_t(\mathbf{e}(u))$;
3. $\mathbf{e}$ *preserves the relation* child*:* $\forall (u, v) \in Edges_{\downarrow}(P)$, $(\mathbf{e}(u), \mathbf{e}(v)) \in Edges_t$;
4. $\mathbf{e}$ *preserves the relation* descendant*:* $\forall (u, v) \in Edges_{\Downarrow}(P)$, $(\mathbf{e}(u), \mathbf{e}(v)) \in (Edges_t)^+$,

where $(Edges_t)^+$ is the transitive closure of the relation $Edges_t$.

The notion of model is illustrated in Figure 2.

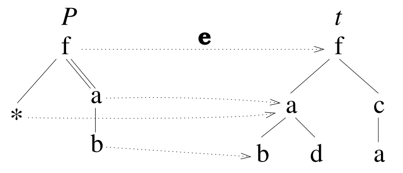

Fig. 2. Pattern P, its model t, and embedding $\mathbf{e}$ from P to t

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.

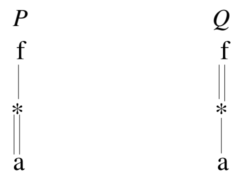

Fig. 3. Equivalent patterns P and Q

Miklau and Suciu prove in [1] 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 $\varphi\colon Nodes_Q \to Nodes_P$, which is:*

- *root and name preserving;*
- *child preserving:*
  $\forall (u,v) \in Edges_{\downarrow}(Q),\ (\varphi(u),\varphi(v)) \in Edges_{\downarrow}(P)$*;*
- *descendant preserving:*
  $\forall (u,v) \in Edges_{\Downarrow}(Q),\ (\varphi(u),\varphi(v)) \in (Edges_{\downarrow}(P) \cup Edges_{\Downarrow}(P))^{+}$.

The authors of [1] 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 $\mathcal{O}(|P||Q|)$, whether there exists a homomorphism from Q to P. Figure 4 shows the patterns P and Q, and the homomorphism φ from Q to P proving that $P \subseteq Q$. Nevertheless, the existence of a homomorphism from Q

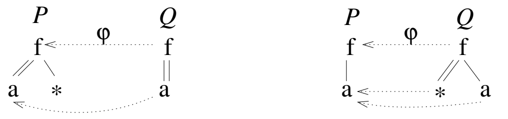

Fig. 4. Homomorphism φ from Q to P proving that $P \subseteq Q$

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 [1]) satisfying $P \subseteq Q$, such that there is no homomorphism from Q to P. Here,

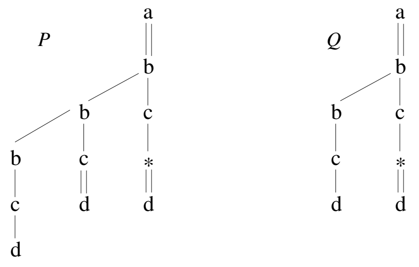

Fig. 5. Patterns s.t. $P \subseteq Q$, but no homomorphism from Q to P

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),

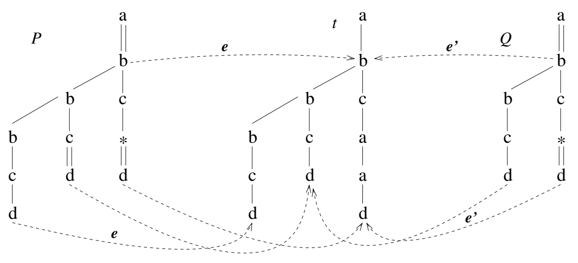

Fig. 6. Model of P (and Q), where $c//d$ is realized by c/d

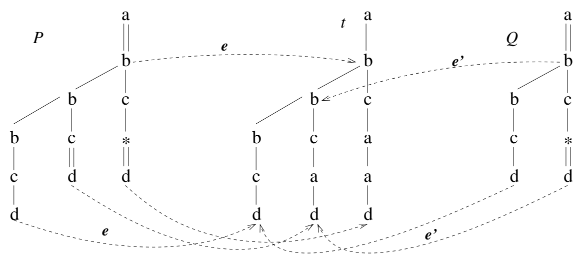

Fig. 7. Model of P (and Q), where $c//d$ is realized by $c/a/d$

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

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 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 Σ as the expressions P derived from the grammar of Table 1, where $\downarrow$ and $\Downarrow$ stand respectively for* `child` *and* `descendant`*, $\omega \in \Sigma \cup \{*\}$, and '$*$' is the* don't–care *symbol of XPath that can replace any σ of Σ.*

This grammar produces precisely the patterns as defined in [7,1]. For instance, the graph P in Figure 8 corresponds to the expression

$$P = a \Downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \Downarrow d\}, \downarrow c \downarrow * \Downarrow d\}$$

M :	$\varepsilon \mid \downarrow\omega \mid \Downarrow\omega \mid MM$	// path
S :	$\emptyset \mid \{MS\} \mid S \cup S$	// set of sibling unrooted terms
P :	ωMS	// patterns

Table 1. Grammar for patterns

derived from the grammar of Table 1.

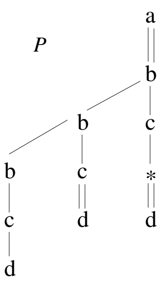

Fig. 8. Pattern

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 $P_1 \vee P_2 \vee \cdots \vee P_n$ 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 $\Sigma \cup \{*\}$), 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 ε, P, or $P_1 \vee \cdots \vee P_n$, will also be called *d–patterns*. A tree t is a model of a d–pattern $P_1 \vee \cdots \vee P_n$ iff t is a model of at least one pattern P_i, for $1 \leq i \leq n$. Definition 2 of pattern containment is extended in a natural way to a d–pattern containment. 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

$$P = f \downarrow * \Downarrow a \qquad \text{and} \qquad Q = f \Downarrow * \downarrow a$$

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 ≥ 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

$$f \downarrow * \downarrow a \quad \vee \quad f \downarrow * \Downarrow * \downarrow a$$

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 ≥ 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:

$$f \Downarrow * \downarrow a \quad \vee \quad f \Downarrow * \downarrow a,$$

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 $\omega, \omega' \in \Sigma \cup \{*\}$:

1. $S \longrightarrow \emptyset,\ \ M \longrightarrow \varepsilon$ //cut;
2. $M\sigma S \longrightarrow M * S$ //replace any symbol of Σ by the '$*$' of XPath;
3. $\downarrow \omega S \longrightarrow \Downarrow \omega S$ //every child is also a descendant;
4. $\xi\omega\xi'\omega' S \longrightarrow \Downarrow \omega' S$ //ignore an intermediate node;
5. $M\{S_1, S_2\} \longrightarrow \{MS_1, MS_2\}$ //left distributivity;
6. $S \longrightarrow S \cup S'$, where $S \longrightarrow S'$ //add new siblings;
7. $S \cup S_1 \longrightarrow S' \cup S_1$, if $S \longrightarrow S'$ //rewrite some of the siblings;
8. $\Downarrow \omega S \longrightarrow (\downarrow \omega S) \vee (\downarrow * \Downarrow \omega S)$ //case analysis: descendant is either a child or has depth ≥ 2;
9. $\Downarrow \omega S \longrightarrow (\downarrow \omega S) \vee (\Downarrow * \downarrow \omega S)$ //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 $\mathcal{C}\blacklozenge X$) to be the pattern obtained from $\mathcal{C}$ by replacing its hole symbol with the term X; e.g. for the context–pattern $\mathcal{C} = f\{\downarrow a, \Downarrow b\{\Diamond, \downarrow d\}, \Downarrow *\}$, and the unrooted term $X = \Downarrow x\{\downarrow y, \Downarrow z\}$, we get the fill–in:

$$\mathcal{C}\blacklozenge X = f\{\downarrow a, \Downarrow b\{\Downarrow x\{\downarrow y, \Downarrow z\}, \downarrow d\}, \Downarrow *\}.$$

We also suppose that for any context–pattern $\mathcal{C}$ and unrooted terms X and X', the notation $\mathcal{C}\blacklozenge(X \vee X')$ stands for the disjunctive d–pattern $\mathcal{C}\blacklozenge X \ \vee\ \mathcal{C}\blacklozenge X'$.

To rewrite patterns with the rules of $\mathcal{R}$ given above, 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, if there exist a context–pattern $\mathcal{C}$ and two unrooted terms X and X', such that: $P = \mathcal{C}\blacklozenge X$, $Q = \mathcal{C}\blacklozenge X'$, 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. $D_1 \vee D \longrightarrow D_2 \vee D$, if D_1 can be rewritten to D_2 //case rewriting;
11. $P \vee P \vee D \longrightarrow P \vee D$ //consider any given case only once.

Rules 10 and 11 are used as follows: if a d–pattern L is an instance (modulo commutativity) of the LHS of rule 10 or 11, and a d–pattern R is an instance (modulo commutativity) of the RHS of the same rule, then L can be rewritten to R. We will denote by $A \longrightarrow_{\mathcal{R}} B$ the fact that a pattern or a d–pattern A is rewritten in one step to a pattern or a d–pattern B, by using the rules of $\mathcal{R}$.

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 $P \overset{*}{\longrightarrow}_{\mathcal{R}} Q$, 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 $P \overset{*}{\longrightarrow}_{\mathcal{R}} Q$ implies $P \subseteq Q$. Indeed, if $X \longrightarrow X'$ is an instance of one of the rules $1-9$, then for every context–pattern $\mathcal{C}$, we have $\mathcal{C}\blacklozenge X \subseteq \mathcal{C}\blacklozenge X'$; if $L \longrightarrow R$ is an instance of rule 10 or 11, we obviously have $L \subseteq R$.

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 $P \overset{*}{\longrightarrow}_{\mathcal{R}} Q$.*

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

(a) for every node u of Q, we construct a corresponding node u' of P', and we set $name_{P'}(u') = name_P(\varphi(u))$;
(b) we construct a child edge $(u', v') \in Edges_{\downarrow}(P')$, if and only if $(u, v) \in Edges_{\downarrow}(Q)$;
(c) we construct a descendant edge $(u', v') \in Edges_{\Downarrow}(P')$, if and only if $(u, v) \in Edges_{\Downarrow}(Q)$.

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 $u \in Nodes_Q$ and the corresponding node $u' \in Nodes_{P'}$ may be different. Condition (a) implies that: either $name_Q(u) = name_{P'}(u') = name_P(\varphi(u))$, or $name_Q(u) \neq name_{P'}(u') = name_P(\varphi(u))$. In the second case we have (see Definition 3): $name_Q(u) = *$, and $name_{P'}(u') \in \Sigma$, 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 φ;
– 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 $Edges_{\downarrow}(P')$: from condition (b) we know that $(u, v) \in Edges_{\downarrow}(Q)$, thus by Definition 3 we have $(\varphi(u), \varphi(v)) \in Edges_{\downarrow}(P)$ (we have nothing to do with the edge $(\varphi(u), \varphi(v))$ when rewriting P to P');
– case when edge (u', v') is in $Edges_{\Downarrow}(P')$: from condition (c) and Definition 3 we can deduce that there exist $k \geq 1$ and $w_0, \dots w_k \in Nodes_P$, such that: $w_0 = \varphi(u), w_k = \varphi(v)$, and $\forall\ i \in \{0, \dots, k-1\}$ we have $(w_i, w_{i+1}) \in Edges_{\downarrow}(P) \cup Edges_{\Downarrow}(P)$. If $k = 1$ and $(\varphi(u), \varphi(v)) \in Edges_{\downarrow}(P)$, then we can rewrite P to P' using rule 3. If $k \geq 2$, then we use ($k - 1$ times) rule 4 to ignore the nodes $w_1, \dots w_{k-1}$ while rewriting P to P'.

Finally, we obtain $P \overset{*}{\longrightarrow}_{\mathcal{R}} P' \overset{*}{\longrightarrow}_{\mathcal{R}} Q$. □

Note that if P is a tree, we also have the converse of Lemma 1. Indeed, it is sufficient to remark that if $P \overset{*}{\longrightarrow}_{\mathcal{R}} Q$ and P is a tree, then one can rewrite P to Q by using only rules $1-7$; if $X \longrightarrow X'$ is an instance of one of those rules, then for every context–pattern $\mathcal{C}$, there exists a homomorphism from $\mathcal{C}\blacklozenge X'$ to $\mathcal{C}\blacklozenge X$. Of course, in the case when P is a tree, a homomorphism from Q to P is an embedding from the pattern Q to the tree P. The above considerations give us the following characterization:

Remark 1. A tree t is a model of a pattern Q iff $t \overset{*}{\longrightarrow}_{\mathcal{R}} Q$.

By a *homomorphism* from a pattern Q to a d–pattern $D = P_1 \vee \dots \vee P_n$, 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 D, then $D \overset{*}{\longrightarrow}_{\mathcal{R}} Q$.*

Proof. It suffices to remark that rules 10 and 11 imply that a d–pattern $P_1 \vee \dots \vee P_n$ can be rewritten to a pattern Q if and only if, for every $1 \leq i \leq n$, we have $P_i \overset{*}{\longrightarrow}_{\mathcal{R}} Q$. □

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 $P \overset{*}{\longrightarrow}_{\mathcal{R}} D$, such that there exists a homomorphism from Q to D.*

Proof. From the result of Miklau and Suciu ([1]) 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 satisfying the proposition can be constructed by using rules 8 and 9 finitely many times. We know that every model of P is also a model of Q. The idea is to represent all models of P by an equivalent d–pattern $D = P_1 \vee \dots \vee P_n$ 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 $P = a \Downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \Downarrow d\}, \downarrow c \downarrow * \Downarrow d\}$ can be seen as the fill–in

$$a \Downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c\Diamond\}, \downarrow c \downarrow * \Downarrow d\} \ \blacklozenge \ \underline{\Downarrow d}.$$

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

$$\begin{array}{ll} & a \Downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c\Diamond\}, \downarrow c \downarrow * \Downarrow d\} \ \blacklozenge \ \downarrow d \\ \vee & a \Downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c\Diamond\}, \downarrow c \downarrow * \Downarrow d\} \ \blacklozenge \ \downarrow * \Downarrow d. \end{array}$$

We obtain the d–pattern

$$\begin{array}{ll} & a \Downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow d\}, \downarrow c \downarrow * \Downarrow d\} \\ \vee & a \Downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \Downarrow d\}, \downarrow c \downarrow * \Downarrow d\}, \end{array}$$

which can be seen under the form

$$\begin{array}{ll} & a \Downarrow b\{\downarrow b\{\Diamond, \downarrow c \downarrow d\}, \downarrow c \downarrow * \Downarrow d\} \ \blacklozenge \ \underline{\downarrow b \downarrow c \downarrow d} \\ \vee & a \Downarrow b\{\downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \Downarrow d\}, \Diamond\} \ \blacklozenge \ \underline{\downarrow c \downarrow * \Downarrow d}. \end{array}$$

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

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

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

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

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

$$a \Downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \Downarrow d\} \quad \vee \quad a \Downarrow b\{\downarrow b \downarrow c \downarrow d, \downarrow c \downarrow * \Downarrow d\} = Q \vee Q,$$

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 $P = * \Downarrow *$ and $Q = * \downarrow *$, we have $P \subseteq Q$ ($P \xrightarrow{*}_{\mathcal{R}} Q$ using rules $9, 1$), but $P\Diamond \ \blacklozenge \ \downarrow a = * \Downarrow * \downarrow a$ is *not* contained in $Q\Diamond \ \blacklozenge \ \downarrow a = * \downarrow * \downarrow a$: for instance, the tree $t = f \downarrow g \downarrow b \downarrow a$ is a model of $* \Downarrow * \downarrow a$, but not of $* \downarrow * \downarrow a$.

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 [11,12,13,14]). Figure 9 represents three formats of the same document: tree, fully and partially compressed format (see [11] for formal definitions). In the

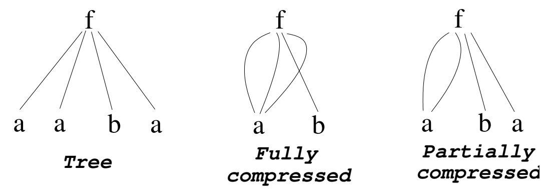

Fig. 9. Tree, fully compressed format, partially compressed format

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 greater then the indexes of non–terminals for parent nodes.

Such normalized grammars are *straightline* in the sense defined in [15], 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:

$$\begin{array}{lll}
X_0 \to f(X_1, X_2, X_3, X_4) & Y_0 \to f(Y_1, Y_1, Y_2, Y_1) & Z_0 \to f(Z_1, Z_1, Z_2, Z_3) \\
X_1 \to a & Y_1 \to a & Z_1 \to a \\
X_2 \to a & Y_2 \to b & Z_2 \to b \\
X_3 \to b & & Z_3 \to a. \\
X_4 \to a & &
\end{array}$$

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 $\downarrow$ or $\Downarrow$, 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 $\downarrow$. 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.

$$\begin{array}{ll} P_0 \to *(\downarrow P_1, \Downarrow P_2) & \qquad X_0 \to f(\downarrow X_2, \downarrow X_1) \\ P_1 \to a & \qquad X_1 \to b(\downarrow X_2) \\ P_2 \to a & \qquad X_2 \to a. \end{array}$$

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

To define an embedding $\mathbf{e}$ from a pattern P to a dag t, we replace the conditions 3 and 4 of Definition 1 respectively by:

3. $\forall u, v \in Nodes_P$, such that $(u, v) \in Edges_{\downarrow}(P)$, there exists an edge going form $\mathbf{e}(u)$ to $\mathbf{e}(v)$ on t;
4. $\forall u, v \in Nodes_P$, such that $(u, v) \in Edges_{\Downarrow}(P)$, there exists a path going from $\mathbf{e}(u)$ to $\mathbf{e}(v)$ in $(Edges_t)^+$.

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.

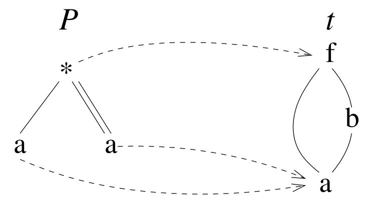

Fig. 11. Pattern P, its compressed model t, and embedding from P to t

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 $\mathcal{R}$:

$$\begin{array}{lll}
X_0 \to f(\downarrow X_2, \downarrow X_1) \xrightarrow{2} & X_0 \to *(\downarrow X_2, \downarrow X_1) \xrightarrow{1} & X_0 \to *(\downarrow X_2) \xrightarrow{6} \\
X_1 \to b(\downarrow X_2) & X_1 \to b(\downarrow X_2) \xrightarrow{1} & \\
X_2 \to a & X_2 \to a & X_2 \to a
\end{array}$$

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 $X_1 \to b(\downarrow X_2)$, since it has become unproductive (there is no more production having X_1 on their right hand sides).

$$\begin{array}{llll}
X_0 \to *(\downarrow X_2, \downarrow X_2') \xrightarrow{3} & X_0 \to *(\downarrow X_2, \Downarrow X_2') & \approx & P_0 \to *(\downarrow P_1, \Downarrow P_2) \\
X_2 \to a & X_2 \to a & \approx & P_1 \to a \\
X_2' \to a & X_2' \to a & \approx & P_2 \to a.
\end{array}$$

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-

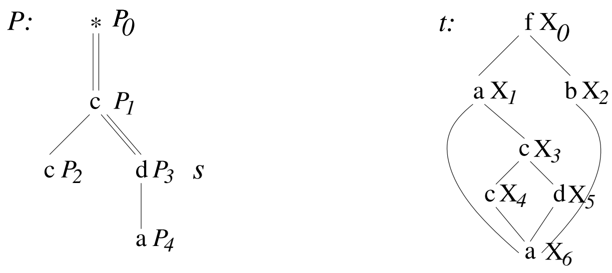

Fig. 12. Unary pattern P and its compressed model t

resenting the query $P =$ /∗//c[./c]//d[./a], and the compressed document t, given in Figure 12. The corresponding SLR grammars G_P and G_t are respectively:

$$\begin{array}{ll}
P_0 \to *(\Downarrow P_1) & X_0 \to f(\downarrow X_1, \downarrow X_2) \\
P_1 \to c(\downarrow P_2, \Downarrow P_3) & X_2 \to b(\downarrow X_6) \\
P_2 \to c & X_1 \to a(\downarrow X_6, \downarrow X_3) \\
P_3(s) \to d(\downarrow P_4) & X_3 \to c(\downarrow X_4, \downarrow X_5) \\
P_4 \to a & X_4 \to c(\downarrow X_6) \\
 & X_5 \to d(\downarrow X_6) \\
 & X_6 \to a.
\end{array}$$

The non–terminal P_3 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 $\mathcal{R}$. The non–terminal of G_t which will be rewritten to the selecting non–terminal P_3 of G_P, will represent an answer for P on t. We illustrate this reasoning below:

$$\begin{array}{lll}
X_0 \to f(\downarrow X_1, \downarrow X_2) \xrightarrow{1} & X_0 \to f(\downarrow X_1) \xrightarrow{2} & X_0 \to *(\downarrow X_1) \xrightarrow{4} \\
X_2 \to b(\downarrow X_6) & & \\
X_1 \to a(\downarrow X_6, \downarrow X_3) \xrightarrow{1} & X_1 \to a(\downarrow X_3) & X_1 \to a(\downarrow X_3) \\
X_3 \to c(\downarrow X_4, \downarrow X_5) & X_3 \to c(\downarrow X_4, \downarrow X_5) \xrightarrow{3} & X_3 \to c(\downarrow X_4, \Downarrow X_5) \\
X_4 \to c(\downarrow X_6) \xrightarrow{1} & X_4 \to c & X_4 \to c \\
X_5 \to d(\downarrow X_6) & X_5 \to d(\downarrow X_6) & X_5 \to d(\downarrow X_6) \\
X_6 \to a & X_6 \to a & X_6 \to a
\end{array}$$

$$\begin{array}{lcl}
X_0 \to *(\Downarrow X_3) & \approx & P_0 \to *(\Downarrow P_1) \\
X_3 \to c(\downarrow X_4, \Downarrow X_5) & \approx & P_1 \to c(\downarrow P_2, \Downarrow P_3) \\
X_4 \to c & \approx & P_2 \to c \\
X_5 \to d(\downarrow X_6) & \approx & P_3(s) \to d(\downarrow P_4) \\
X_6 \to a & \approx & P_4 \to a,
\end{array}$$

We have obtained an SLR grammar, which is (up to non–terminal renaming) the SLR grammar G_P for P. The non–terminal X_5 of G_t has been rewritten to the non–terminal P_3, thus the node represented by X_5 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 ascendant axes (cf.[11]). 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 ([16]), 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 staightline context–free grammars (SLCF, [15]), 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

1. Miklau, G., Suciu, D.: Containment and Equivalence for a Fragment of XPath. J. ACM **51**(1) (2004) 2–45
2. Neven, F., Schwentick, T.: XPath Containment in the Presence of Disjunction, DTDs, and Variables. In Calvanese, D., Lenzerini, M., Motwani, R., eds.: ICDT '03. Volume 2572 of LNCS., Springer (2003) 312–326
3. W3C: XML Path Language. `http://www.w3.org/TR/xpath` (1999)
4. Wood, P.T.: On the Equivalence of XML Patterns. In Lloyd, J.W., Dahl, V., Furbach, U., Kerber, M., Lau, K.K., Palamidessi, C., Pereira, L.M., Sagiv, Y., Stuckey, P.J., eds.: CL '00. Volume 1861 of LNCS., London, UK, Springer (2000) 1152–1166
5. Schwentick, T.: XPath Query Containment. SIGMOD Rec. **33**(1) (2004) 101–109
6. Gottlob, G., Koch, C., Pichler, R.: Efficient Algorithms for Processing XPath Queries. ACM Trans. Database Syst. **30**(2) (2005) 444–491
7. Flesca, S., Furfaro, F., Masciari, E.: On the Minimization of XPath Queries. J. ACM **55**(1) (2008)
8. Amer-Yahia, S., Cho, S., Lakshmanan, L.V.S., Srivastava, D.: Tree Pattern Query Minimization. VLDB J. **11**(4) (2002) 315–331
9. Kimelfeld, B., Sagiv, Y.: Revisiting Redundancy and Minimization in an XPath Fragment. In: EDBT '08: Proceedings of the 11th International Conference on Extending Database Technology, New York, USA, ACM (2008) 61–72
10. Ramanan, P.: Efficient Algorithms for Minimizing Tree Pattern Queries. In: SIGMOD '02: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, New York, USA, ACM (2002) 299–309
11. Fila, B., Anantharaman, S.: Automata for Positive Core XPath Queries on Compressed Documents. In Herman, M., Voronkov, A., eds.: LPAR '06. Volume 4246 of LNAI., Springer (2006) 467–481
12. Buneman, P., Grohe, M., Koch, C.: Path Queries on Compressed XML. In: VLDB '03, Morgan Kaufmann (2003) 141–152
13. Marx, M.: XPath and Modal Logics of Finite DAG's. In Cialdea-Mayer, M., Pirri, F., eds.: TABLEAUX '03. Volume 2796 of LNAI., Springer (2003) 150–164

14. Frick, M., Grohe, M., Koch, C.: Query Evaluation on Compressed Trees (Extended Abstract). In: LICS '03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science, Washington, DC, USA, IEEE Computer Society (2003) 188
15. Busatto, G., Lohrey, M., Maneth, S.: Efficient Memory Representation of XML Document Trees. Inf. Syst. **33**(4-5) (2008) 456–474
16. Niehren, J., Planque, L., Talbot, J.M., Tison, S.: N–ary Queries by Tree Automata. In Bierman, G.M., Koch, C., eds.: DPL '05. Volume 3774 of LNCS., Springer (2005) 217–231