A Study of Positive XPath with Parent/Child Navigation

Yuqing Wu
Dirk Van Gucht
Indiana University, Bloomington
yuqwu@indiana.edu
vgucht@cs.indiana.edu

Marc Gyssens
Hasselt University & Transnational University of Limburg
marc.gyssens@uhasselt.be

Jan Paredaens
University of Antwerp
jan.paredaens@ua.ac.be

ABSTRACT

We study the expressiveness of Positive XPath with parent/child navigation, denoted XPath$^+$, from two angles. First, we establish that XPath$^+$ is equivalent in expressive power to some of its sub-fragments as well as to the class of tree queries, a sub-class of the first-order conjunctive queries defined over label, parent, and child predicates. The translation algorithm from tree queries to XPath$^+$ yields a simple normal form for XPath$^+$ queries. Using this normal form, we can effectively partition an XPath$^+$ query into subqueries that can be expressed in a very small sub-fragment of XPath$^+$ for which efficient evaluation strategies are available. Second, we characterize the expressiveness of XPath$^+$ in terms of its ability to distinguish nodes in a document. We show that two such nodes cannot be distinguished if and only if the paths from the root of the documents to these nodes have equal length and corresponding nodes on these paths are bisimilar.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*query languages*

General Terms

Languages, Theory

Keywords

XPath, expressiveness, normal form

1. INTRODUCTION

XPath is a language for navigation in XML documents [4, 5]. In the node-set semantics, an XPath expression associates to each node of a document the set of nodes to which can be navigated. In the path semantics, an XPath expression is viewed as a query that associates to each document a binary relation on its nodes representing all paths in the document defined by that expression. We will use the term *query expressiveness* when we refer to the ability of XPath to express queries. Alternatively, the expressiveness of XPath can be investigated by characterizing its ability to distinguish nodes in a document using well-chosen expressions. We will use the term *resolution expressiveness* to refer to this notion of expressiveness. [9]. Apart from expressiveness issues, the study of efficient evaluation of XPath expressions has drawn a lot of attention. [7, 8, 10]

The W3C standard for XPath is complex and unwieldy. Therefore, researchers have introduced clean algebraic and logical abstractions in order to study this language formally. To gain additional insight, this study does not only involve full XPath, but also many of its fragments [7, 2, 13, 12, 11, 9, 3]. Consequently, the literature on the formal aspects of XPath has become very extensive.[1]

In this paper, we study XPath$^+$, which is a small fragment of CoreXPath [7, 11]. XPath$^+$ allows parent/child navigation and disallows union and difference. More concretely, its expressions are

$$\begin{aligned} E \quad ::= \quad & \emptyset \,|\, \varepsilon \,|\, \hat{\ell} \,|\, \downarrow \,|\, \uparrow \,| \\ & E;E \,|\, E[E] \,|\, \Pi_1(E) \,|\, \Pi_2(E) \,|\, E^{-1} \,|\, E \cap E, \end{aligned}$$

where the primitives $\emptyset$, ε, $\hat{\ell}$, $\downarrow$, $\uparrow$ respectively denote the empty set, the empty path, label selection, child edges, and parent edges, and the operators ;, [ ], Π_1, Π_2, $^{-1}$, and $\cap$ denote composition, predication (semi-join), first projection, second projection, inverse, and intersection of binary relations of nodes. It can be seen as the algebra $\mathcal{X}^{\uparrow}_{[\,]}$ introduced by Benedikt et al. [2] (without union), augmented with first and second projection, inverse, and intersection.

Consider an XPath$^+$ expression E and a document D. Under the path semantics, $E(D)$ is a binary relation over the nodes of D. In the node-set semantics, also known as the navigational semantics, we can view $E(D)$ as the function that associates to each node m in D the set of nodes $E(D)(m) = \{n \mid (m, n) \in E(D)\}$.

The node-set semantics of the language $\mathcal{X}^{\uparrow}_{[\,]}$ is well-studied. The following are some of its known properties:

1. *Under node-set semantics*, $\mathcal{X}^{\uparrow}_{[\,]}$ and the first-order conjunctive queries with two free variables defined over label and child predicates have the same query expressiveness [2];

2. *Under node-set semantics*, $\mathcal{X}^{\uparrow}_{[\,]}$ and the unary tree queries with child edges have the same query expressiveness [3];

[1]For a good survey, see [3].

and

3. *Under node-set semantics*, every query that can be expressed in $\mathcal{X}^{\uparrow}_{[\,]}$ is equivalent to a unique, minimum-size unary tree query which can be computed using tree reduction techniques. [14, 1]

We can now state the main results of the present paper. Let XPath$^+(\cap)$ denote the fragment of XPath$^+$ defined by

$$E ::= \emptyset \,|\, \varepsilon \,|\, \hat{\ell} \,|\, \downarrow \,|\, \uparrow \,|\, E;E \,|\, E \cap E,$$

and let XPath$^+(\Pi_1, \Pi_2)$ denote the fragment of XPath$^+$ defined by

$$E ::= \emptyset \,|\, \varepsilon \,|\, \hat{\ell} \,|\, \downarrow \,|\, \uparrow \,|\, E;E \,|\, \Pi_1(E) \,|\, \Pi_2(E).$$

Then, relative to the query expressiveness of XPath$^+$, we show that

1. *Under path semantics*, XPath$^+$, XPath$^+(\cap)$, and XPath$^+(\Pi_1, \Pi_2)$ have the same query expressiveness;

2. *Under path semantics*, XPath$^+$ has the same query expressiveness as tree-queries with *two* return nodes;

3. *Under path semantics*, every query that can be expressed in XPath$^+$ is equivalent to a unique, minimum-size tree query with *two* return nodes which can be computed using tree reduction techniques.

Relative to the resolution expressiveness of XPath$^+$, we show that two nodes in a document cannot be distinguished by XPath$^+$ expressions if and only if the paths from the root in the document to these nodes have equal length and corresponding nodes on these paths are bisimilar. Notice that the query expressiveness results mentioned above simplify the proof of this result. Conversely, our resolution expressiveness results yield a tool to prove the minimization result mentioned above.

Finally, by examining the translation algorithm from tree queries to XPath$^+(\Pi_1, \Pi_2)$ expressions, we show that each XPath$^+$ expressions can be transformed into a simple normal form. This normal form effectively partitions an XPath$^+$ query into sub-expressions that can be expressed in a very small fragment of XPath$^+$ for which efficient evaluation strategies are available. [6]

2. DOCUMENTS

Throughout the paper, we assume an infinitely enumerable set $\mathcal{L}$ of *labels*.

Definition 2.1. *A* document D *is a labeled tree* (V, Ed, λ), *with* V *the set of nodes,* $Ed \subseteq V \times V$ *the set of edges, and* $\lambda : V \to \mathcal{L}$ *a node-labeling function.*

For two arbitrary nodes m and n in an document D, there is a unique path from m to n if we ignore the orientation of the edges. The unique node on this path that is an ancestor of both m and n will henceforth be denoted top(m, n).

Example 2.2. *Figure 1 shows an example of a document that will be used throughout the paper. Notice that, in this document,* $top(n_8, n_{12}) = n_4$.

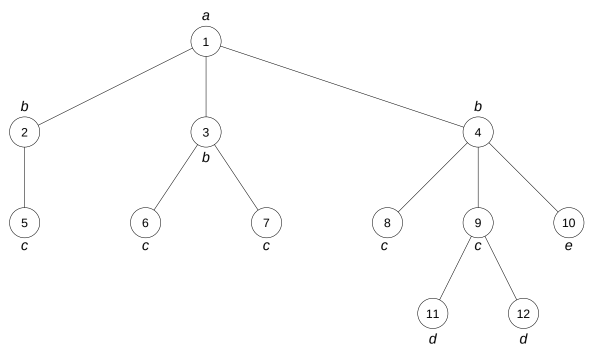

Figure 1: An example document.

3. POSITIVE XPATH WITH PARENT/CHILD NAVIGATION

Here, we give the formal definition of XPath$^+$ and its path semantics.

Definition 3.1. *XPath*$^+$ *is an algebra which consists of the primitives* $\emptyset$, ε, $\hat{\ell}$ ($\ell \in \mathcal{L}$), $\downarrow$, *and* $\uparrow$, *together with the operations composition* ($E_1; E_2$), *predication* ($E_1[E_2]$), *first projection* ($\Pi_1(E)$), *second projection* ($\Pi_2(E)$), *inverse* (E^{-1}), *and intersection* ($E_1 \cap E_2$). (E, E_1, *and* E_2 *represent XPath*$^+$ *expressions.)*

Given a document specification $D = (V, Ed, \lambda)$, *the* path semantics *of an XPath*$^+$ *expression is a binary relation over* V, *defined as follows:*

- $\emptyset(D) = \emptyset$;
- $\varepsilon(D) = \{(n, n) \mid n \in V\}$;
- $\hat{\ell}(D) = \{(n, n) \mid n \in V \text{ and } \lambda(n) = \ell\}$;
- $\downarrow(D) = Ed$;
- $\uparrow(D) = Ed^{-1}$;
- $E_1; E_2(D) = \pi_{1,4}(\sigma_{2=3}(E_1(D) \times E_2(D)))$;
- $E_1[E_2](D) = \pi_{1,2}(\sigma_{2=3}(E_1(D) \times E_2(D)))$;
- $\Pi_1(E)(D) = \pi_1(E(D))$;
- $\Pi_2(E)(D) = \pi_2(E(D))$;
- $E^{-1}(D) = E(D)^{-1}$; *and*
- $E_1 \cap E_2(D) = E_1(D) \cap E_2(D)$,

where E, E_1, *and* E_2 *represent XPath*$^+$ *expressions.*

A special XPath$^+$ expression will be used throughout this paper.

Definition 3.2. *Let* D *be a document and let* m *and* n *be arbitrary nodes of* D. *Then, the* signature *of the pair* (m, n) *(denoted* $sig(m, n)$*) is the XPath*$^+$ *expression* $\uparrow^k; \downarrow^\ell$, *with* k *the length of the path between* $top(m, n)$ *and* m, ℓ *the length of the path between* $top(m, n)$ *and* n.[2]

[2]Here, exponentiation denotes repeated composition. For an XPath$^+$ expression E, E^0 denotes ε.

The signature of a pair of nodes of a document can be seen as a description of the unique path connecting these nodes, but also as an expression that can be applied to the document under consideration. We shall often exploit this duality.

For two pairs of nodes (m_1, n_1) and (m_2, n_2) in a document D, we say that $\mathrm{sig}(m_1, n_1) \geq \mathrm{sig}(m_2, n_2)$ if (m_2, n_2) is in $\mathrm{sig}(m_1, n_1)(D)$. Clearly, $\mathrm{sig}(m_1, n_1) = \mathrm{sig}(m_2, n_2)$ if and only if $\mathrm{sig}(m_1, n_1) \geq \mathrm{sig}(m_2, n_2)$ and $\mathrm{sig}(m_2, n_2) \geq \mathrm{sig}(m_1, n_1)$.

EXAMPLE 3.3. *In the document in Figure 1, sig*(n_5, n_6) = *sig*$(n_7, n_8) = \uparrow^2; \downarrow^2$*; whereas sig*$(n_6, n_7) = \uparrow; \downarrow$*. Notice that sig*$(n_5, n_6) \geq$ *sig*(n_8, n_9)*, but not the other way around.*

By restricting the operators allowed in expressions, several sub-algebras of XPath$^+$ can be defined. The following two are of special interest to us:

- XPath$^+(\cap)$ is the sub-algebra of XPath$^+$ where, besides the primitives and the composition operator, only intersection is allowed; and
- XPath$^+(\Pi_1, \Pi_2)$ is the sub-algebra of XPath$^+$ where, besides the primitives and the composition operator, only the first and second projections are allowed.

4. TREE QUERIES WITH PARENT/CHILD NAVIGATION

DEFINITION 4.1. *A* tree query *with parent/child navigation is a 3-tuple* (T, s, d)*, with* T *a labeled tree, and* s *and* d *nodes of* T*, called the source and destination nodes. The nodes of* T *are either labeled with a symbol of* $\mathcal{L}$ *or with a* wildcard *denoted "∗," which is assumed not to be in* $\mathcal{L}$*.*

Two symbols of $\mathcal{L} \cup \{*\}$ are called *compatible* if they are either equal or one of them is a wildcard. For two compatible symbols ℓ_1 and ℓ_2, we define $\ell_1 + \ell_2$ to be ℓ_1 if ℓ_1 is not a wildcard, and ℓ_2 otherwise.

Let $P = ((V', Ed', \lambda'), s, d)$ be a tree query, and let $D = (V, Ed, \lambda)$ be a document. A *containment mapping* of P in D is a mapping $h : V' \to V$ such that

1. for all tree query nodes $m', n' \in V'$, $(m', n') \in Ed'$ implies that $(h(m'), h(n')) \in Ed$; and
2. for every tree query node $m' \in V'$, $\lambda'(m') \in \mathcal{L}$ implies that $\lambda(h(m')) = \lambda'(m')$.

Observe that a containment mapping is in fact a homomorphism with respect to the child and label predicates if the tree query does not contain wildcards.

We can now define the semantics of a tree query.

DEFINITION 4.2. *Let* $P = (T, s, d)$ *be a tree query, and let* D *be a document. The semantics of* P *given* D*, denoted* $P(D)$*, is defined as the set*

$$\{(h(s), h(d)) \mid h \textit{ is a containment mapping of } P \textit{ in } D\}.$$

To the set of all tree queries, we add "$\emptyset$," which returns the empty set on any given document. The resulting set of expressions is denoted $\mathbf{T}$.

EXAMPLE 4.3. *Figure 2 shows an example of a tree query that will be used throughout the paper. The semantics of this tree query given the document in Figure 1 is the following set of pairs of nodes of that document:* $\{(n_9, n_{11}), (n_9, n_{12})\}$*.*

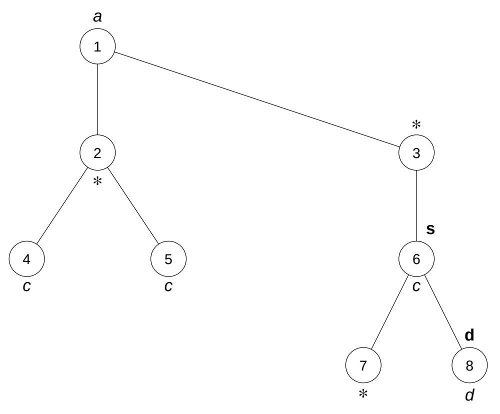

Figure 2: An example tree query.

5. EQUIVALENCE OF QUERY LANGUAGES

In this section, we show that, under path semantics, XPath$^+$, $\mathbf{T}$, XPath$^+(\cap)$, and XPath$^+(\Pi_1, \Pi_2)$ have the same query expressiveness by exhibiting a translation algorithm that translates an expression in one language to an equivalent expression in one of the other languages. In this regard, we want to point out that the equivalence between XPath$^+$ and $\mathbf{T}$ can easily be derived from results on $\mathcal{X}^{\uparrow}_{[]}$ described in the Introduction. For the sake of completeness, however, we present the entire translation algorithm, in particular because this algorithm is important for Section 8 where a normal form for XPath$^+$ queries will be defined.

LEMMA 5.1. *The query languages XPath*$^+$ *and XPath*$^+(\cap)$ *are equivalent in expressive power, and there exists an algorithm translating an arbitrary XPath*$^+$ *expression into an equivalent XPath*$^+(\cap)$ *expression.*

PROOF. First, we give a recursive algorithm to translate an XPath$^+$ expression E to an equivalent expression $\tau_1(E)$ in the sub-algebra XPath$^+(\cap, ^{-1})$ where, besides the primitives and the composition operator, only intersection and inverse are allowed. The translation algorithm consists of the following rewriting rules:

(1) $\tau_1(E) = E$ if E is a primitive;
(2) $\tau_1(E_1; E_2) = \tau_1(E_1); \tau_1(E_2)$;
(3) $\tau_1(E_1[E_2]) = \tau_1(E_1); ((\tau_1(E_2); \tau_1(E_2)^{-1}) \cap \varepsilon)$;
(4) $\tau_1(\Pi_1(E)) = (\tau_1(E); \tau_1(E)^{-1}) \cap \varepsilon$;
(5) $\tau_1(\Pi_2(E)) = (\tau_1(E)^{-1}; \tau_1(E)) \cap \varepsilon$;
(6) $\tau_1(E^{-1}) = \tau_1(E)^{-1}$; and
(7) $\tau_1(E_1 \cap E_2) = \tau_1(E_1) \cap \tau_1(E_2)$.

Finally, we give a recursive algorithm that, given an XPath$^+(\cap)$ expression E, translates E^{-1} to an equivalent expression $\tau_2(E)$ in XPath$^+(\cap)$:

(1) $\tau_2(\emptyset) = \emptyset$; $\tau_2(\varepsilon) = \varepsilon$; $\tau_2(\hat{\ell}) = \hat{\ell}$ $(\ell \in \mathcal{L})$;
(2) $\tau_2(\uparrow) = \downarrow$; $\tau_2(\downarrow) = \uparrow$;
(3) $\tau_2(E_1; E_2) = \tau_2(E_2); \tau_2(E_1)$;
(4) $\tau_2(E^{-1}) = E$; and
(5) $\tau_2(E_1 \cap E_2) = \tau_2(E_1) \cap \tau_2(E_2)$.

□

EXAMPLE 5.2. *Consider the XPath*$^+$ *expression* $\uparrow[\uparrow]$*. Using the translation algorithm in the proof of Lemma 5.1, we*

Algorithm Merge1

```
Input: two disjoint labeled trees $T_1 = (V_1, Ed_1, \lambda_1)$ and
    $T_2 = (V_2, Ed_2, \lambda_2)$;
    nodes $m_1 \in V_1$ and $m_2 \in V_2$.

Output: a labeled tree or "$\emptyset$."

Method: let $\delta = \min(\text{depth}(m_1, T_1), \text{depth}(m_2, T_2))$;
    for $k = 0, \ldots, \delta$
        if the level-$k$ ancestors of $m_1$ and $m_2$ have
            incompatible labels, return $\emptyset$;
    for $k = 0, \ldots, \delta$
        merge the level-$k$ ancestors $m_1^k$ of $m_1$ and $m_2^k$ of $m_2$
            into a node labeled $\lambda_1(m_1^k) + \lambda_2(m_2^k)$;
    return the resulting labeled tree.
```

Figure 3: The algorithm *Merge1*.

Algorithm Merge2

```
Input: a labeled tree $T = (V, Ed, \lambda)$ and nodes $m_1, m_2 \in V$;

Output: a labeled tree or "$\emptyset$."

Method: if $\text{depth}(m_1, T) \neq \text{depth}(m_2, T)$ return $\emptyset$
        else let $\delta = \text{depth}(m_1, T) = \text{depth}(m_2, T)$;
    for $k = 0, \ldots, \delta$
        if the level-$k$ ancestors of $m_1$ and $m_2$ have
            incompatible labels, return $\emptyset$;
    for $k = 0, \ldots, \delta$
        merge the level-$k$ ancestors $m_1^k$ of $m_1$ and $m_2^k$ of $m_2$
            into a node labeled $\lambda_1(m_1^k) + \lambda_2(m_2^k)$;
    return the resulting labeled tree.
```

Figure 4: The algorithm *Merge2*.

find that this expression is equivalent to $\uparrow; (\uparrow; \downarrow \cap \varepsilon)$, an expression of XPath$^+(\cap)$.

LEMMA 5.3. *The query language **T** is at least as expressive as XPath$^+(\cap)$, and there exists an algorithm translating an arbitrary XPath$^+(\cap)$ expression into an equivalent expression of **T** (i.e., a tree query or "$\emptyset$").*

PROOF. It is straightforward to translate the primitives to expressions of **T**. Now, let E_1 and E_2 be XPath$^+(\cap)$ expressions for which P_1 and P_2 are the equivalent expressions in **T**. If one of P_1 or P_2 equals "$\emptyset$," then both $E_1; E_2$ and $E_1 \cap E_2$ are translated into "$\emptyset$." Otherwise, let $P_1 = (T_1, s_1, d_1)$ and $P_2 = (T_2, s_2, d_2)$ be the two tree queries under consideration.

1. *Translation of composition.* Apply the algorithm *Merge1* (Figure 3) to the labeled trees T_1 and T_2 and the nodes d_1 and s_2. If the result is "$\emptyset$," then this is also the translation of $E_1; E_2$. Otherwise, let T be the returned labeled tree. Then $E_1; E_2$ is translated into the tree query $P = (T, s_1, d_2)$.

2. *Translation of intersection.* First, apply the algorithm *Merge1* to the labeled trees T_1 and T_2 and the nodes s_1 and s_2. If the result is "$\emptyset$," then this is also the translation of $E_1 \cap E_2$. Otherwise let T_{int} be the labeled tree returned by *Merge1*. Next, apply the algorithm *Merge2* (Figure 4) to the labeled tree T_{int} and the nodes d_1 and d_2. If the result is "$\emptyset$," then this is also the translation of $E_1 \cap E_2$. Otherwise, let T be the labeled tree returned by *Merge2*. Then $E_1 \cap E_2$ is translated into the tree query $P = (T, s, d)$, where s is the node that resulted from merging s_1 and s_2, and d is the node that resulted from merging d_1 and d_2.

□

EXAMPLE 5.4. *Figure 5 shows how the XPath$^+(\cap)$ expression $\uparrow; (\uparrow; \downarrow \cap \varepsilon)$ is translated into a tree query.*

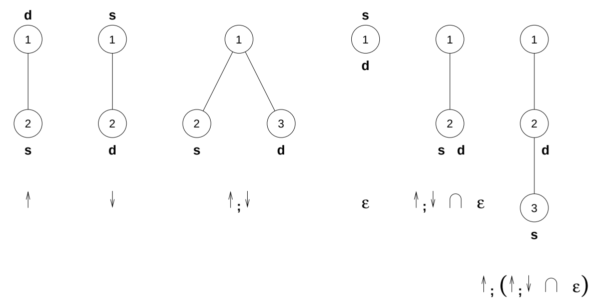

Figure 5: A translation from XPath$^+(\cap)$ into T. All nodes in the tree queries are assumed to be labeled by a wildcard.

LEMMA 5.5. *The query language XPath$^+(\Pi_1, \Pi_2)$ is at least as expressive as **T**, and there exists an algorithm translating an arbitrary **T** expression into an equivalent expression of XPath$^+(\Pi_1, \Pi_2)$.*

PROOF. Clearly, "$\emptyset$" is translated into "$\emptyset$." Also, the translation of tree queries with at most two nodes is straightforward. We therefore consider a tree query $P = (T, s, d)$ with at least three nodes, and give a recursive translation algorithm. We distinguish the following cases. They are schematically illustrated in Figure 6.

1. *s is neither a leaf nor the root and d is not a strict descendant of s.* Let T_1 be the sub-tree of T rooted at s and T_2 the result of removing all strict descendants of s from T. Then, the translation of P is the composition of the translations of $P_1 = (T_1, s, s)$ and $P_2 = (T_2, s, d)$.

2. *s is a leaf and $d \neq s$.* Let p be the parent of s. Let T_1 be the tree consisting of the single edge (p, s), and T_2 be the result of removing s from T. Then, the translation of P is the composition of the translations of $P_1 = (T_1, s, p)$ and $P_2 = (T_2, p, d)$.

3. *d is neither a leaf nor the root and s is not a descendant of d.* Analogous to Case 1.

4. *d is a leaf and $s \neq d$.* Analogous to Case 2.

5. *$s = d$ is a leaf.* Let r be the root of T. Then, the translation of P is the second projection of the translation of $P' = (T, r, d)$.

6. *$s = d$ is the root.* Let c be any child of s. Then, the translation of P is the first projection of the translation of $P' = (T, s, c)$.

□

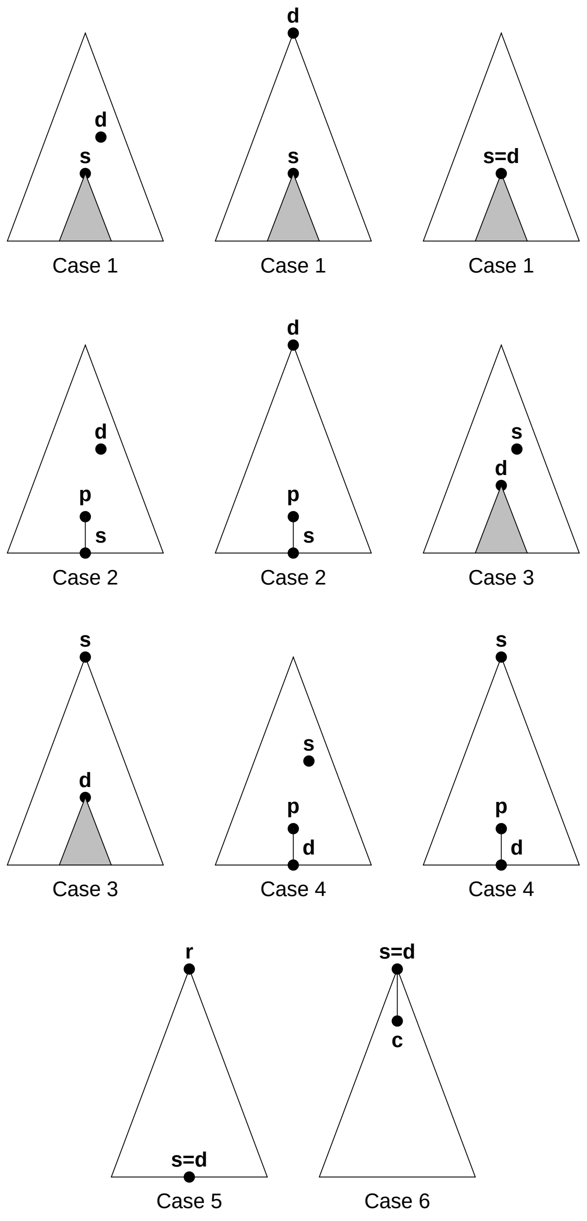

Figure 6: Schematic illustration of the various cases distinguished in the proof of Lemma 5.5. Notice that the node p may be equal to d if p is the parent of s, and vice-versa.

EXAMPLE 5.6. *Consider again the rightmost tree query in Figure 5. By applying rules 2 and 5 in the proof of Lemma 5.5, this tree query is translated into* $\uparrow; \Pi_2(\downarrow)$, *an expression of XPath*$^+(\Pi_1, \Pi_2)$.

Notice that Examples 5.2, 5.4, and 5.6, yield a translation of the XPath$^+$ *expression* $\uparrow [\uparrow]$ *into the XPath*$^+(\Pi_1, \Pi_2)$ *expression* $\uparrow; \Pi_2(\downarrow)$.

We can now summarize Lemmas 5.1, 5.3, and 5.5.

THEOREM 5.7. *The four query languages XPath*$^+$, ***T***, *XPath*$^+(\cap)$, *and XPath*$^+(\Pi_1, \Pi_2)$ *are all equivalent in expressive power, and there exist translation algorithms between any two of them.*

6. RESOLUTION EXPRESSIVENESS

In this section, we establish that two nodes in a document cannot be distinguished by an XPath$^+$ expression if and only if the paths from the root of that document to these nodes have equal length, and corresponding nodes on that paths are bisimilar, a property that has been called *1-equivalence* in [9]. The proof has the same structure as the proofs of similar properties for other fragments of XPath in [9].

We first make precise what we mean by indistinguishability of nodes.

DEFINITION 6.1. *Let m_1 and m_2 be nodes of a document D.*

- *m_1 and m_2 are* expression-related *(denoted $m_1 \geq_{exp} m_2$) if, for each XPath$^+$ expression E, $E(D)(m_1) \neq \emptyset$ implies $E(D)(m_2) \neq \emptyset$.*
- *m_1 and m_2 are* expression-equivalent *(denoted $m_1 \equiv_{exp} m_2$) if $m_1 \geq_{exp} m_2$ and $m_2 \geq_{exp} m_1$.*

In fragments of XPath containing a negation operator (e.g., set difference), expression relatedness implies expression equivalence. Unfortunately, this is not the case for XPath$^+$. Clearly, expression equivalence is an equivalence relation, but expression relatedness is not.

We now turn to the formal definition of 1-equivalence.

DEFINITION 6.2. *Let m_1 and m_2 be nodes of a document D. Then m_1 and m_2 are* downward 1-related *(denoted $m_1 \geq^1_\downarrow m_2$) if and only if*

1. *$\lambda(m_1) = \lambda(m_2)$; and*
2. *for each child n_1 of m_1, there exists a child n_2 of m_2 such that $n_1 \geq^1_\downarrow n_2$.*

Obviously, two nodes are bisimilar (called *downward 1-equivalent* in [9]) if they are downward 1-related in both directions.

DEFINITION 6.3. *Let m_1 and m_2 be nodes of a document D. Then m_1 and m_2 are* 1-related *(denoted $m_1 \geq^1 m_2$) if*

1. *$m_1 \geq^1_\downarrow m_2$; and*
2. *if m_1 is not the root, and p_1 is the parent of m_1, then m_2 is not the root and $p_1 \geq^1 p_2$, with p_2 the parent of m_2.*

We finally define 1-equivalence.

DEFINITION 6.4. *Let m_1 and m_2 be nodes of a document D. Then m_1 and m_2 are* 1-equivalent *(denoted $m_1 \equiv^1 m_2$) if and only if $m_1 \geq^1 m_2$ and $m_2 \geq^1 m_1$.*

EXAMPLE 6.5. *Consider again the document in Figure 1. We have $n_2 \equiv^1 n_3$, $n_5 \equiv^1 n_6 \equiv^1 n_7$, and $n_{11} \equiv^1 n_{12}$. In addition, we have $n_2 \geq^1 n_4$, $n_3 \geq^1 n_4$, $n_5 \geq^1 n_8$, $n_5 \geq^1 n_9$, $n_6 \geq^1 n_8$, $n_6 \geq^1 n_9$, $n_7 \geq^1 n_8$, and $n_7 \geq^1 n_9$. None of these latter relationships can be strengthened to 1-equivalence.*

For the purpose of abbreviation, we extend 1-relatedness to pairs of nodes, as follows. Let m_1, m_2, n_1, and n_2 be nodes of a document D. We say that $(m_1, n_1) \geq^1 (m_2, n_2)$ whenever $m_1 \geq^1 m_2$, $n_1 \geq^1 n_2$, and $\mathrm{sig}(m_1, n_1) \geq \mathrm{sig}(m_2, n_2)$.

EXAMPLE 6.6. *In the document in Figure 1, $n_5 \geq^1 n_8$, $n_6 \geq^1 n_9$, and $sig(n_5, n_6) \geq sig(n_8, n_9)$. Hence, $(n_5, n_6) \geq^1 (n_8, n_9)$.*

The following technical properties of 1-relatedness of pairs of nodes are useful in the sequel.

LEMMA 6.7. *Let m_1, m_2, n_1, and n_2 be nodes of a document D such that $(m_1, n_1) \geq^1 (m_2, n_2)$. Now, let p_1 be any node on the path between m_1 and n_1. Let p_2 be the unique node that is either on the path between m_2 and n_2 or an ancestor of $top(m_2, n_2)$ such that $sig(m_1, p_1) \geq sig(m_2, p_2)$*[3]*. Then $p_1 \geq^1 p_2$.*

PROOF. The nodes p_1 and p_2 are either ancestors of respectively m_1 and m_2 or ancestors of respectively n_1 and n_2. From Definition 6.3, it readily follows that same-level ancestors of 1-related nodes are themselves 1-related. □

EXAMPLE 6.8. *Continuing with Example 6.6, Lemma 6.7 states that, in this particular case, $n_5 \geq^1 n_8$, $n_2 \geq^1 n_4$, $n_1 \geq^1 n_1$, $n_3 \geq^1 n_4$, and $n_6 \geq^1 n_9$, which is indeed the case.*

LEMMA 6.9. *Let m_1, m_2, and n_1 be nodes of a document D such that $m_1 \geq^1 m_2$. Then there exists a node n_2 such that $(m_1, n_1) \geq^1 (m_2, n_2)$.*

PROOF. A straightforward inductive proof can be given for the case that m_2 is either an ancestor or a descendant of m_1. Using that $\mathrm{top}(m_1, n_1)$ is an ancestor of m_1 and that n_1 is a descendant of $\mathrm{top}(m_1, n_1)$, we bootstrap the special case to the general case. (Details omitted.) □

LEMMA 6.10. *Let m_1, m_2, n_1, n_2, and p_1 be nodes of a document D such that $(m_1, n_1) \geq^1 (m_2, n_2)$. Then, there exists a node p_2 such that $(m_1, p_1) \geq^1 (m_2, p_2)$ and $(p_1, n_1) \geq^1 (p_2, n_2)$.*

PROOF. By a case analysis considering the possible mutual positions of m_1, n_1, and p_1. (Details omitted.) □

LEMMA 6.11. *Let E be an XPath$^+$ expression, and let m_1, m_2, n_1, and n_2 be nodes of a document D such that $(m_1, n_1) \geq^1 (m_2, n_2)$. If $(m_1, n_1) \in E(D)$, then it is also the case that $(m_2, n_2) \in E(D)$.*

PROOF. By Theorem 5.7, we may assume without loss of generality that E is an XPath$^+(\cap)$ expression. The proof now proceeds by structural induction. For the primitives $\emptyset$, ε, $\hat{\ell}$ ($\ell \in \mathcal{L}$), $\downarrow$, and $\uparrow$, it is straightforward that the lemma holds. This settles the base case. We now turn to the inductive step:

1. $E := E_1; E_2$, with E_1 and E_2 satisfying the lemma. Assume $(m_1, n_1) \in E(D)$. Then there exists a node p_1 such that $(m_1, p_1) \in E_1(D)$ and $(p_1, n_1) \in E_2(D)$. By Lemma 6.10, there exists a node p_2 such that $(m_1, p_1) \geq^1 (m_2, p_2)$ and $(p_1, n_1) \geq^1 (p_2, n_2)$. By the induction hypothesis, $(m_2, p_2) \in E_1(D)$ and $(p_2, n_2) \in E_2(D)$. Thus, $(m_2, n_2) \in E(D)$.

[3]Or, equivalently, $\mathrm{sig}(p_1, n_1) \geq \mathrm{sig}(p_2, n_2)$.

2. $E := E_1 \cap E_2$, with E_1 and E_2 satisfying the lemma. Assume $(m_1, n_1) \in E(D)$. Then, $(m_1, n_1) \in E_1(D)$ and $(m_1, n_1) \in E_2(D)$. Then, by the induction hypothesis, $(m_2, n_2) \in E_1(D)$ and $(m_2, n_2) \in E_2(D)$. Thus, $(m_2, n_2) \in E(D)$.

□

COROLLARY 6.12. *Let E be an XPath$^+$ expression, and let m_1, m_2, and n_1 be nodes of a document D such that $m_1 \geq^1 m_2$ and $(m_1, n_1) \in E(D)$. Then there exists a node n_2 such that $(m_2, n_2) \in E(D)$.*

PROOF. By Lemma 6.9, there exists a node n_2 such that $(m_1, n_1) \geq^1 (m_2, n_2)$. By Lemma 6.11, $(m_2, n_2) \in E(D)$. □

We are now ready to develop an argument that shows that the semantic property of expression equivalence and the syntactic property of 1-equivalence coincide. The argument is divided in the following three lemmas.

LEMMA 6.13. *Let m_1 and m_2 be nodes of a document D. If $m_1 \geq^1 m_2$, then $m_1 \geq_{exp} m_2$.*

PROOF. Assume that $m_1 \geq^1 m_2$. We show that $m_1 \geq_{\mathrm{exp}} m_2$. Thereto, let E be an XPath$^+$ expression. Suppose that $E(D)(m_1) \neq \emptyset$. Hence, there exists a node n_1 such that $(m_1, n_1) \in E(D)$. By Corollary 6.12, there exists a node n_2 such that $(m_2, n_2) \in E(D)$. Hence, $E(D)(m_2) \neq \emptyset$. □

LEMMA 6.14. *Let m_1 and m_2 be nodes of a document D. If $m_1 \geq_{exp} m_2$, then $m_1 \geq^1_{\downarrow} m_2$.*

PROOF. If $m_1 \geq_{\mathrm{exp}} m_2$, then $\lambda(m_1) = \lambda(m_2)$. Otherwise, $\widehat{\lambda(m_1)}(D)(m_1) \neq \emptyset$ and $\widehat{\lambda(m_1)}(D)(m_2) = \emptyset$, a contradiction. Hence, condition 1 of Definition 6.2 is satisfied.

Now, we prove that $m_1 \geq_{\mathrm{exp}} m_2$ implies $m_1 \geq^1_{\downarrow} m_2$ by induction on the height of the sub-tree rooted at m_1.

If m_1 is a leaf, it suffices to prove that $\lambda(m_1) = \lambda(m_2)$, which, by the above, is the case. This settles the base case.

In the inductive case, let n_1^1 be a child of m_1. Then m_2 cannot be a leaf, for, otherwise, $\downarrow(D)(m_1) \neq \emptyset$ and $\downarrow(D)(m_2) = \emptyset$, a contradiction with $m_1 \geq_{\mathrm{exp}} m_2$. Let $n_2^1, \ldots, n_2^\ell$ be all children of m_2. Suppose that, for all i, $1 \leq i \leq \ell$, $n_1^1 \not\geq_{\mathrm{exp}} n_2^i$. Hence, for all i, $1 \leq i \leq \ell$, there exists an XPath$^+$ expression E_i such that $E_i(D)(n_1^1) \neq \emptyset$ and $E_i(D)(n_2^i) = \emptyset$. Let $F := \varepsilon[\varepsilon[E_1] \cap \ldots \cap \varepsilon[E_i]]$. Then $\downarrow; F(D)(m_1) \neq \emptyset$ and $\downarrow; F(D)(m_2) = \emptyset$, a contradiction with $m_1 \geq_{\mathrm{exp}} m_2$. Hence, there exists a child n_2^j of m_2, $1 \leq j \leq \ell$, such that $n_1^1 \geq_{\mathrm{exp}} n_2^j$. By the induction hypothesis, $n_1^1 \geq^1_{\downarrow} n_2^j$. By Definition 6.2, $m_1 \geq^1_{\downarrow} m_2$. □

LEMMA 6.15. *Let m_1 and m_2 be nodes of a document D. If $m_1 \geq_{exp} m_2$, then $m_1 \geq^1 m_2$.*

PROOF. By induction on the depth of m_1.

If m_1 is the root, then m_2 must also be the root. Indeed, let δ be the depth of D. Then $\downarrow^\delta (D)(m_1) \neq \emptyset$. Hence, $\downarrow^\delta (D)(m_2) \neq \emptyset$. This is only possible is m_2 is also the root. Clearly, a node is 1-related to itself.

Thus, suppose that m_1 is not the root. Hence $\uparrow(D)(m_1) \neq \emptyset$, so $\uparrow(D)(m_2) \neq \emptyset$. It follows that m_2 cannot be the root either. Thus, let p_1 be the parent of m_1 and p_2 the parent of m_2. If $p_1 \not\geq_{\mathrm{exp}} p_2$, there exists an XPath$^+$ expression E such that $E(D)(p_1) \neq \emptyset$ and $E(D)(p_2) = \emptyset$. Obviously, then $\uparrow; E(D)(m_1) \neq \emptyset$ and $\uparrow; E(D)(m_2) = \emptyset$, a contradiction with $m_1 \geq_{\text{exp}} m_2$. Thus, $p_1 \geq_{\text{exp}} p_2$. By the induction hypothesis, $p_1 \geq^1 p_2$. By Lemma 6.14, we also know that $m_1 \geq^1_{\downarrow} m_2$. By Definition 6.3, $m_1 \geq^1 m_2$. □

Lemmas 6.13 and 6.15 now immediately yield

THEOREM 6.16. *Let m_1 and m_2 be nodes of a document D. Then, $m_1 \geq_{exp} m_2$ if and only if $m_1 \geq^1 m_2$.*

Theorem 6.16 now immediately yields the desired result.

COROLLARY 6.17. *Let m_1 and m_2 be nodes of a document D. Then, $m_1 \equiv_{exp} m_2$ if and only if $m_1 \equiv^1 m_2$.*

7. MINIMIZATION OF TREE QUERIES

In this section, we show that results on containment and minimization of tree queries can easily be derived using the theory developed in Section 6.

First, we extend the notion of containment mapping (Section 4). Thereto, let $P_1 = ((V_1, Ed_1, \lambda_1), s_1, d_1)$ and $P_2 = ((V_2, Ed_2, \lambda_2), s_2, d_2)$ be tree queries. A *query containment mapping* of P_1 in P_2 is a mapping $h : V_1 \to V_2$ such that

1. for all nodes $m_1, n_1 \in V_1$, $(m_1, n_1) \in Ed_1$ implies that $(h(m_1), h(n_1)) \in Ed_2$;
2. for every node $m_1 \in V_1$, $\lambda_1(m_1) \in \mathcal{L}$ implies that $\lambda_2(h(m_1)) = \lambda_1(m_1)$; and
3. $h(s_1) = s_2$ and $h(d_1) = d_2$.

The following is straightforward:

PROPOSITION 7.1. *Let P_1 and P_2 be tree queries. Then P_2 is contained in P_1 if and only if there is a query containment mapping of P_1 in P_2.*

We now extend the notion of 1-relatedness from nodes of a document to nodes of a pattern by replacing the condition $\lambda(m_1) = \lambda(m_2)$ in Definition 6.2 by the three conditions (1) $\lambda(m_1) + \lambda(m_2) = \lambda(m_2)$; (2) $m_1 = s$ implies $m_2 = s$; and (3) $m_1 = d$ implies $m_2 = d$. We shall then say that m_1 and m_2 are 1-$*$-related and denote this by $m_1 \geq^1_* m_2$.

If m_1 and m_2 are 1-$*$-related in both directions, i.e., $m_1 \geq^1_* m_2$ and $m_2 \geq^1_* m_1$, we say that m_1 and m_2 are 1-$*$-equivalent and denote this by $m_1 \equiv^1_* m_2$.

Now, let P be a tree query. We define the *first reduction* of P, denoted $\text{red}_1(P)$, to be the tree query obtained from P by merging 1-$*$-equivalent nodes.

EXAMPLE 7.2. *The tree query in Figure 7,* left, *is the first reduction of the tree query in Figure 2.*

LEMMA 7.3. *A tree query and its first reduction are equivalent.*

PROOF. Obviously, the merging of nodes of P that yields $\text{red}_1(P)$ defines a query containment mapping of P in $\text{red}_1(P)$. Furthermore, let m_1, m_2, and n_1 be nodes of P such that $m_1 \equiv^1 m_2$ and (m_1, n_1) is an edge of P. It follows from Definition 6.4 that there exists a node n_2 in P such that $n_1 \equiv^1 n_2$ and (m_2, n_2) is an edge of P. Using this property, it can easily be seen that there also exists a containment mapping in the other direction. □

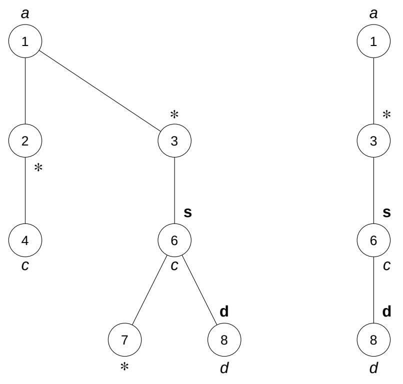

Figure 7: The first reduction (*left*) and the second reduction (*right*) of the tree query in Figure 2.

EXAMPLE 7.4. *Continuing with Example 7.2, observe that the mapping from the nodes of the first reduction to the nodes of the original tree query defined by the node numbers is a query containment mapping. In the other direction, the mapping from the nodes of the original tree query to the nodes of the first reduction defined by $1 \mapsto 1$, $2 \mapsto 2$, $3 \mapsto 3$, $4 \mapsto 4$, $5 \mapsto 4$, $6 \mapsto 6$, $7 \mapsto 7$, and $8 \mapsto 8$ is also a query containment mapping. We may thus conclude that both tree queries are indeed equivalent.*

For P a tree query, we define the *second reduction* of P, denoted $\text{red}_2(P)$, to be the tree query by deleting from $\text{red}_1(P)$ in a top-down fashion every node m_1 for which there exists *another* node m_2 such that $m_1 \geq^1_* m_2$.

Notice that the purpose of doing the reduction in two steps is to ensure that that the graph of the relation "$\geq^1_*$" is acyclic.

EXAMPLE 7.5. *The tree query in Figure 7,* right, *is the second reduction of the tree query in Figure 2.*

LEMMA 7.6. *A tree query and its second reduction are equivalent.*

PROOF. Since the second reduction $\text{red}_2(P)$ of a tree query P is a sub-tree of its first reduction $\text{red}_1(P)$, there is obviously a query containment mapping from $\text{red}_2(P)$ into $\text{red}_1(P)$. To see that there is also a query containment mapping in the other direction, we note that the algorithm to construct $\text{red}_2(P)$ from $\text{red}_1(P)$ actually yields such a mapping. □

EXAMPLE 7.7. *Continuing with Example 7.5, observe that the mapping from the nodes of the second reduction to the nodes of the first reduction defined by the node numbers is a query containment mapping. In the other direction, the mapping from the nodes of the first reduction to the nodes of the second reduction defined by $1 \mapsto 1$, $2 \mapsto 3$, $3 \mapsto 3$, $4 \mapsto 6$, $6 \mapsto 6$, $7 \mapsto 8$, and $8 \mapsto 8$ is also a query containment mapping. We may thus conclude that both tree queries are indeed equivalent.*

We can now show the following.

THEOREM 7.8. *Let P be a tree query. Every (with respect to number of nodes) minimal tree query equivalent to P is isomorphic to $\text{red}_2(P)$.*

PROOF. Let Q be a tree query with at most as many nodes as $\text{red}_2(P)$. By Proposition 7.1 and Lemmas 7.3 and 7.6, there exist query containment mappings h_{PQ} and h_{QP} from $\text{red}_2(P)$ into Q and Q into $\text{red}_2(P)$, respectively. Then, $h = h_{QP} \circ h_{PQ}$ is a query containment mapping from $\text{red}_2(P)$ into itself. Let m be an arbitrary node of $\text{red}_2(P)$. Then, clearly, $m \geq^1_* h(m)$. Since $\text{red}_2(P)$ is reduced, $m = h(m)$. Now, h can only be the identity if Q has at least as many nodes as $\text{red}_2(P)$. Hence, Q and $\text{red}_2(P)$ have the same number of nodes. Consequently, h_{QP} and h_{QP} are bijections that are each others inverses. This is only possible if both functions always map wildcard-labeled nodes to wildcard-labeled nodes. We may thus conclude that Q and $\text{red}_2(P)$ are isomorphic. $\square$

8. NORMAL FORMS AND EFFICIENT QUERY EVALUATION

When we revisit Section 5, where the translation from XPath^+ to $\text{XPath}^+(\Pi_1, \Pi_2)$ is described, and, in particular, the proof of Lemma 5.5, where the transition from $\mathbf{T}$ to $\text{XPath}^+(\Pi_1, \Pi_2)$ is explicited, we observe that the result of the translation has the following general form:

$$\emptyset \,|\, \varepsilon \,|\, (\alpha \uparrow)^* (\Pi_2((\alpha \downarrow)^+))^? \alpha (\downarrow \alpha)^*$$

with $\alpha ::= \hat{\ell}^? (\Pi_1(\downarrow \alpha))^*$ ($\ell \in \mathcal{L}$). In the expression above, concatenation must be interpreted as composition.

In the most general case, the expression above is of the form $(\alpha; \uparrow)^*; \beta; (\downarrow; \alpha)^*$. If we interpret this expression in the context of tree queries, then α is a filter condition on the nodes on the ascending and descending branches of the path between the source and the destination nodes, whereas β is a filter condition of the top node of this path.

We say that $\text{XPath}^+(\Pi_1, \Pi_2)$ expressions of this form are XPath^+ expressions in *normal form*.

Normalization is often a critical step in rule-based query optimization. It serves the purpose of unifying queries with the same semantics, detecting containment among sub-queries, and establishing the foundation for cost-based query optimization, in which evaluation plans are to be generated. Rewriting queries using sub-queries for which efficient evaluation plans are known is a practice used extensively in relational database systems. An example of such a rewriting rule is pushing down selections for the selection conditions on which indices are available. It is natural to extend these principle and procedures to XML query processing and optimization.

In [6], the authors established the existence of efficient index-only evaluation plans for a downward fragment of XPath in which the first projection is allowed. When we examine our normal form for XPath^+ queries more closely, we see that the sub-queries corresponding to the filter expressions α belongs this fragment. From similar results in the same paper, it also follows that sub-queries corresponding to larger sub-formulas of the form $(\alpha \uparrow)^*$, $(\alpha \downarrow)^*$, and $(\downarrow \alpha)^*$ can also be evaluated efficiently with index-only plans. In summary, the normal form for XPath^+ queries established in the present paper in combination with the results in [6] yields an efficient index-only evaluation plan for such queries.

Finally, XPath^+ queries can be regarded as the canonical building blocks for more general XPath queries, to be compared with select-project-join queries in the relational algebra.

9. REFERENCES

[1] S. Amer-Yahia, S. Cho, L.V.S. Lakshmanan, and D. Srivastava. Tree pattern query minimization. *VLDB J.*, 11(4):315–331, 2002.

[2] M. Benedikt, W. Fan, and G. M. Kuper. Structural properties of XPath fragments. *Theor. Comput. Sci.*, 336(1):3–31, 2005.

[3] M. Benedikt and C. Koch. XPath leashed. *ACM Computing Surveys*, to appear, 2007.

[4] J. Clark and S. DeRose (eds.). XML path language (XPath) version 1.0. `http://www.w3.org/TR/XPATH`.

[5] A. Berglund et al. (eds.). XML Path Language (XPath) 2.0. `http://www.w3.org/TR/xpath20`.

[6] G.H. Fletcher, D. Van Gucht, Y. Wu, M. Gyssens, S. Brenes, and J. Paredaens. A methodology for coupling fragments of XPath with structural indexes for XML documents. In *DBPL*, pages 48–65, 2007.

[7] G. Gottlob, C. Koch, and R. Pichler. Efficient Algorithms for Processing XPath Queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005.

[8] G. Gou and R. Chirkova. Efficiently querying large XML data repositories: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(10):1381–1403, 2007.

[9] M. Gyssens, J. Paredaens, D. Van Gucht, and G.H.L. Fletcher. Structural characterizations of the semantics of XPath as navigation tool on a document. In S. Vansummeren, editor, *PODS*, pages 318–327. ACM, 2006.

[10] C. Koch. Processing queries on tree-structured data efficiently. In *ACM PODS*, pages 213–224, 2006.

[11] M. Marx. Conditional XPath. *ACM Trans. Database Syst.*, 30(4):929–959, 2005.

[12] M. Marx and M. de Rijke. Semantic characterizations of navigational XPath. *SIGMOD Record*, 34(2):41–46, 2005.

[13] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004.

[14] P. T. Wood. Minimising simple XPath expressions. In *WebDB*, pages 13–18, 2001.