TECHNICAL REPORT NO. 639

An XPath Algebraic Characterization of A(k) and P(k) Indices with Applications to Query Processing

by

George H.L. Fletcher, Dirk Van Gucht, Yuqing Wu,

Marc Gyssens, and Jan Paredaens

October 2006

COMPUTER SCIENCE DEPARTMENT

INDIANA UNIVERSITY

Bloomington, Indiana 47405-4101

An XPath Algebraic Characterization of $A(k)$ and $P(k)$ Indices with Applications to Query Processing

George H.L. Fletcher
Indiana University, Bloomington
gefletch@cs.indiana.edu

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

Yuqing Wu
Indiana University, Bloomington
yuqwu@indiana.edu

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

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

Abstract

Recent studies have proposed structural summary techniques for path query evaluation on semi-structured data sources. One major line of this research has been the introduction of the DataGuide, 1-index, 2-index, and A(k) indices, and subsequent investigations and generalizations. Another recent study has considered structural characterizations of fragments of XPath, the standard path navigation language for XML documents. In this paper we provide a new perspective on XPath query processing which brings together these two areas of research on structural indices and query languages. In particular, we give a precise characterization of the A(k) and P(k) indices in terms of certain algebraic fragments of XPath. With an eye towards applying this result to XPath query processing, we (1) show how expressions in these fragments can be evaluated directly on the corresponding indices; (2) develop a labeling scheme for A(k) and P(k) partition blocks, using algebraic expressions; and (3) leverage these results to develop general techniques for making effective use of A(k) and P(k) indices for important practical classes of XPath.

1 Introduction

XML (eXtensible Markup Language) [1] provides a standard data format that is flexible enough to be customized for various domains. An increasing number of applications use XML as their data model or as the format for exchanging data among heterogeneous data repositories for the purpose of collaboration, data integration, and information sharing.

XQuery is currently the most popular XML query language [3]. The fundamental building block of XQuery is XPath [4]. XPath expressions are used to specify small node-labeled trees which match portions of the hierarchical XML data. How to support efficient access to XML data using XPath continues to be a critical research problem in this domain.

In XPath query evaluation, indices similar to those used in relational database systems – namely, value indices on tags and text values – are first used, together with structural join algorithms [2, 18, 20]. This approach turns out to be simple and efficient. However, the structural containment relationships native to XML data are not directly captured by value indices.

To directly capture the structural information of XML data, a family of XML indices has been introduced. DataGuide [5] was the first to be proposed, followed by the 1-index [12], which is based on a notion of bisimulation among nodes in an XML document. These indices can be used to evaluate path expression accurately without accessing the original data graph. Milo and Suciu [12] also introduced the 2-index and T-index, based on similarity of pairs (vectors) of nodes. Unfortunately, these and other early structural indices tend to be too large for practical use because they typically maintain too fine-grained structural information about the document [8, 16].

To remedy this, Kaushik et al. introduced the $A(k)$-index [10] which uses a notion of bisimilarity on nodes relativized to paths of length k from nodes. This captures localized structural information of a document, and can support path expressions of length up to k. Focusing just on local similarity, the $A(k)$-index can be substantially smaller than the 1-index and others.

Several works have investigated maintenance and tuning of the $A(k)$ indices. The $D(k)$-index [15] and $M(k)$-index [7] extend the $A(k)$-index to adapt to query workload. Yi et al. [19] developed update techniques for the $A(k)$-

Index	Description	Index Nodes	Labeling	Query Evaluation
DataGuide [5]	Every unique label path of a source appear exactly once in the index graph.		Can not use incoming path for node labeling.	Target node set reachable by an XPath expression is the superset of the result.
Strong DataGuide [5]	Distinguishes nodes that are reachable by different paths.	Nodes that share the same incoming path up to the root	Labeled by the unique incoming path up to the root	Can be used to directly answer linear XPath query of any length
1-index [12]	Distinguishes nodes that are reachable by different paths.	Nodes that share the same incoming path up to the root	Labeled by the unique incoming path up to the root	
2-index[12] T-index [12]	Distinguishes pairs (vectors) of nodes that shares the same set of paths.	Pairs (vectors) of nodes that share the same paths		Can be used to directly answer queries that match a given template.
$A(k)$-index [10]	Distinguishes nodes with incoming path up to length k.	Nodes that share the same incoming path, up to length k		Can be used to evaluate linear XPath expression directly when local similarity is sufficient. Need verification when the XPath query expression is longer than k.
$D(k)$-index [15] $M(k)$-index [7]	Uses local index to adapt to query workload	Nodes that shares the same incoming path, according to local similarity.		

Table 1. Comparison Among Structural Indices for Semi-Structured Data.

index and 1-index. Finally, the integrated use of structural and values indices has been explored [9], and there have also been investigations on covering indices [8, 16] and index selection [14, 17]. The characteristics of many of these structural indices are summarized in Table 1.

The introduction of structural indices for XML data has lead to significant improvements in the performance of XPath expression evaluation. Furthermore, great advances have been made on the theoretical and practical underpinnings of expression processing [11]. However, to date there lacks a general framework for investigating some of the most fundamental questions about using indices in expression evaluation. Such questions include:

1. For which class of XPath expressions is an index ideally suited?
2. Then, for this class, how are its expressions optimally evaluated with the index?
3. Can the answers to these questions be bootstrapped to provide general techniques for evaluation of arbitrary XPath expressions?

In this paper we directly address these questions for the $A(k)$ indices, as well as for a new family of localized structural indices on node pairs which we introduce, called the $P(k)$-indices. To answer question (1) for these indices, we follow a general approach to providing structural characterizations of fragments of XPath recently proposed by Gyssens et al. [6]. In particular, we give for the first time a precise characterization of the family of $A(k)$-indices in terms of certain algebraic fragments of XPath (Section 3). We then extend this result to give a precise algebraic characterization of the $P(k)$-indices (Section 3). These couplings of indices with languages hinge on the observation that index blocks can be linked to equivalence classes under a notion of node or path indistinguishability relative to an XPath fragment. To answer question (2) we show how expressions in these fragments can be evaluated directly in terms of the $A(k)$ and $P(k)$ indices (Sections 3 and 4). Finally, to address question (3), we leverage our results on (1) and (2) to develop techniques for making effective use of $A(k)$ and $P(k)$ indices for important practical classes of XPath (Section 5). These results identify a new theoretical methodology for the study of the interaction between structural indices and the evaluation of XPath expressions. Furthermore, the nature of this methodology is such that it sheds light on techniques with immediate practical application.

2 Preliminary Notions

2.1 The XPath-Algebra

We first introduce the XML data model and algebraization of the navigational core of XPath, following the presentation of Gyssens et al. [6]. In this paper, *documents* are finite unordered node-labeled trees. More formally, a document D is a 4-tuple (V, Ed, r, λ), with V the finite set of nodes, $Ed \subseteq V \times V$ the set of edges, $r \in V$ the root, and $\lambda \colon V \to \mathcal{L}$ a node-labeling function into a countably infinite set of labels $\mathcal{L}$. A visualization of a fragment of a document is given in Figure 1. Useful notions of height and distance in a document are defined as follows.

Definition 2.1. *Let $D = (V, Ed, r, \lambda)$ be a document and $n, m \in V$. The* distance *from n to m in D,* $\mathbf{distance}(n, m)$*, is the length of the path from n to m in D. The* height *of D,* $\mathbf{height}(D)$*, is the length of the longest path in D starting from the root r. In other words,* $\mathbf{height}(D) = \max\{\mathbf{distance}(r, n) \mid n \in V\}$.

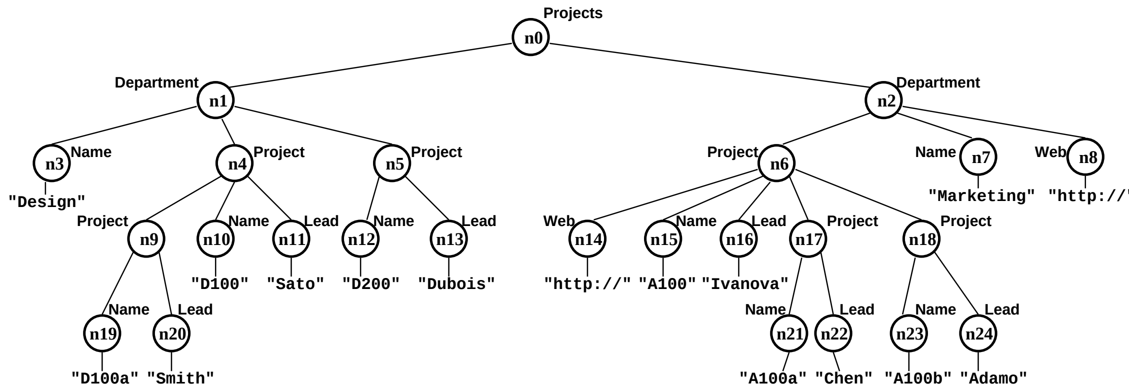


Figure 1. Fragment of an XML document. For reference, non-leaf nodes are given unique IDs.

We next present an algebraization of the logical navigational core of XPath.

Definition 2.2. *The* XPath-algebra *consists of the primitives* $\varepsilon, \emptyset, \downarrow, \uparrow$, *and* ℓ *together with the operations on expressions* $E_1 \diamond E_2$, $E_1[E_2]$, $E_1 \cup E_2$, $E_1 \cap E_2$, *and* $E_1 - E_2$.[1] *Given a document* $D = (V, Ed, r, \lambda)$, *the* semantics *of an XPath-algebra expression* E *on* D, *denoted* $E(D)$, *is always a binary relation over* V. *The semantics for each operation is given in Table 2.*

$$
\begin{aligned}
\varepsilon(D) &= \{(m,m) \mid m \in V\} \\
\emptyset(D) &= \emptyset \\
\downarrow(D) &= Ed \\
\uparrow(D) &= Ed^{-1} \\
\ell(D) &= \{(m,m) \mid m \in V \text{ and } \lambda(m) = \ell\} \\
E_1 \cup E_2(D) &= E_1(D) \cup E_2(D) \\
E_1 \cap E_2(D) &= E_1(D) \cap E_2(D) \\
E_1 - E_2(D) &= E_1(D) - E_2(D) \\
E_1 \diamond E_2(D) &= \{(m,n) \mid \exists w\colon (m,w) \in E_1(D) \\
&\qquad\qquad \& \, (w,n) \in E_2(D)\} \\
E_1[E_2](D) &= \{(m,n) \in E_1(D) \\
&\qquad\qquad \mid \exists w\colon (n,w) \in E_2(D)\}.
\end{aligned}
$$

Table 2. XPath-Algebra Semantics

The XPath-algebra semantics reflects a "global" perspective of expressions being evaluated on an entire document. There is also a "local" semantic perspective, in which expressions are viewed as working on a particular node, as follows.

[1] Since we are concerned in this study with reasoning on a fixed document of which the height is known, it is not necessary to introduce ancestor or descendant axes.

Definition 2.3. *Let* E *be an XPath-algebra expression and let* $D = (V, Ed, r, \lambda)$ *be a document. For* $m \in V$, $E(D)(m) = \{n \in V \mid (m,n) \in E(D)\}$.

To illustrate the XPath-algebra, the XPath query for retrieving all departments with websites on the document of Figure 1,

$$/\texttt{Projects}/\texttt{Department}[./\texttt{Web}]$$

can be formulated in the XPath-algebra with the expression

$$\texttt{Projects} \diamond \downarrow \diamond \texttt{Department}[\downarrow \diamond \texttt{Web}]$$

evaluated "locally" at the root node n_0. As further illustrations of XPath-algebra expressions, consider:

E_1. "Retrieve all department names in projects."

$$\texttt{Projects} \diamond \downarrow \diamond \texttt{Department} \diamond \downarrow \diamond \texttt{Name}$$

E_2. "Retrieve the leaders of projects that have websites."

$$\texttt{Department} \diamond \downarrow \diamond \texttt{Project}[\downarrow \diamond \texttt{Web}] \diamond \downarrow \diamond \texttt{Lead}$$

E_3. "Retrieve the leaders of projects that do not have websites."

$$\texttt{Department} \diamond \downarrow \diamond \texttt{Project} \diamond \downarrow \diamond \texttt{Lead} - E_2$$

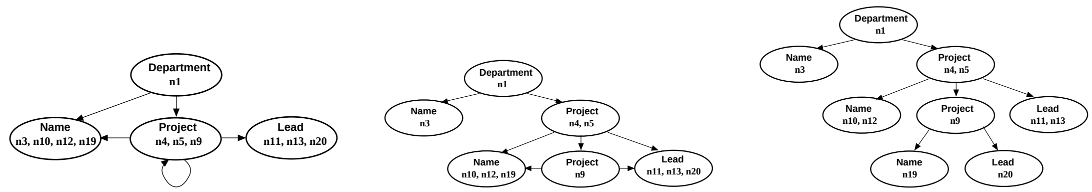
E_4. "Retrieve the names of all departments that have a website and a project with a website."

$$\texttt{Projects} \diamond \downarrow \diamond \texttt{Department}[\downarrow \diamond \texttt{Web}] \\ [\downarrow \diamond \texttt{Project} \diamond \downarrow \diamond \texttt{Web}] \diamond \downarrow \diamond \texttt{Name}$$

E_5. "Retrieve all projects which are sub-projects of projects with a website."

$$\texttt{Project}[\uparrow \diamond \texttt{Project} \diamond \downarrow \diamond \texttt{Web}]$$

Recall that in the "global" semantics of the XPath-algebra, expressions evaluate to pairs of satisfying nodes. For example, $E_1(D) = \{(n_0, n_3), (n_0, n_7)\}$ and $E_5(D) = \{(n_{17}, n_{17}), (n_{18}, n_{18})\}$.



$A(0)$ $A(1)$ $A(2)$

Figure 2. Full $A(k)$ indices for the "Design" Department subtree in the document of Figure 1.

2.2 $A(k)$ Indexes

Kaushik et al. have proposed the family of $A(k)$-indexes for graph-structured data, based on a notion of node bisimilarity [10]. An $A(k)$-index is built on a partitioning of the nodes of a semi-structured document. The definition of the equivalence relation $\equiv_{A(k)}$, specialized to XML documents, which induces this partition is as follows.

Definition 2.4. *Let $D = (V, Ed, r, \lambda)$ be a document, $n_1, n_2 \in V$, and $k \in \mathbb{N}$. We say n_1 and n_2 are* $A(k)$-equivalent in *D, denoted $n_1 \equiv_{A(k)} n_2$, if*

1. *$\lambda(n_1) = \lambda(n_2)$; and*
2. *if $k \geq 1$ then*
   - *(a) n_1 has a parent in D if and only if n_2 has a parent in D; and*
   - *(b) if n_1 has parent p_1 and n_2 has parent p_2, then $p_1 \equiv_{A(k-1)} p_2$.*

$A(k)$ equivalence defines partition blocks on the nodes $n \in V$ of a document $D = (V, Ed, r, \lambda)$,

$$[n]_k = \{n' \mid n' \in V \;\&\; n \equiv_{A(k)} n'\}$$

and the full $A(k)$-index is built directly on the collection of these partition blocks, $D/A(k) = \{[n]_k \mid n \in V\}$. We refer to this partition as simply the $A(k)$-index.

Following Kaushik et al. [10], a full $A(k)$-index can be visualized as a graph wherein each node represents a partition block and an edge exists from a node A to a node B if an edge existed in the original document from a data node in A to a data node in B. This construction is illustrated in Figure 2 on the Design Department subtree of the document of Figure 1.

We observe the following basic property of $A(k)$ equivalence, which follows by a simple induction on k.

Proposition 2.5. *Let $D = (V, Ed, r, \lambda)$ be a document, let $k \in \mathbb{N}$, and let n_1, $n_2 \in V$. If $n_1 \equiv_{A(k+1)} n_2$, then $n_1 \equiv_{A(k)} n_2$.*

At first sight, we have defined an infinite class of equivalence relations on D. However, there is no point to consider $A(k)$ equivalence beyond $k = \mathbf{height}(D)$, as can also be seen by an inductive argument.

Proposition 2.6. *Let $D = (V, Ed, r, \lambda)$ be a document, and let $k \geq \mathbf{height}(D)$. Then $A(k)$ equivalence coincides with $A(\mathbf{height}(D))$ equivalence.*

We now extend $A(k)$ equivalence as follows to capture upward paths to the root in a document.

Definition 2.7. *Let $D = (V, Ed, r, \lambda)$ be a document and $n_1, n_2 \in V$. We say n_1 and n_2 are* $A(\infty)$-equivalent in *D, denoted $n_1 \equiv_{A(\infty)} n_2$, if*

1. *$\lambda(n_1) = \lambda(n_2)$; and*
2. *n_1 has a parent in D if and only if n_2 has a parent in D; and*
3. *if n_1 has parent p_1 and n_2 has parent p_2, then $p_1 \equiv_{A(\infty)} p_2$.*

Not surprisingly, we have the following.

Proposition 2.8. *Let $D = (V, Ed, r, \lambda)$ be a document. Then $A(\infty)$ equivalence coincides with $A(\mathbf{height}(D))$ equivalence.*

Note that $A(\infty)$ equivalence on nodes induces the 1-index proposed by Milo and Suciu [12] and the strong DataGuide of Goldman and Widom [5].

2.3 $P(k)$ Indexes

We next generalize $A(k)$ equivalence (Definition 2.4) to pairs of nodes.

Definition 2.9. *Let $D = (V, Ed, r, \lambda)$ be a document, $k \in \mathbb{N}$, let m_1, n_1, m_2, $n_2 \in V$ such that m_1 is an ancestor of n_1 and m_2 is an ancestor of n_2. Then, (n_1, m_1) and (n_2, m_2) are* $P(k)$ equivalent, *denoted $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$, if*

1. *$\mathbf{distance}(n_1, m_1) = \mathbf{distance}(n_2, m_2)$; and*
2. *$n_1 \equiv_{A(k)} n_2$.*

Note that $P(k)$ equivalence on pairs of nodes induces a "localized" version of the 2-index proposed by Milo and Suciu [12], just as $A(k)$ equivalence on nodes induces a "localized" version of the 1-index [10].

Consider the subtree D' in the document of Figure 1 rooted at n_4. As an illustration of the $P(k)$-index (i.e., the partition set $D/P(k)$ induced by $P(k)$ equivalence on a document D) the $P(0)$-index of D' is:

$$\begin{aligned} D'/P(0) \quad = \quad & \{[(n_{10}, n_{10}), (n_{19}, n_{19})], \\ & [(n_{10}, n_4), (n_{19}, n_9)], [(n_{19}, n_4)], \\ & [(n_4, n_4), (n_9, n_9)], [(n_9, n_4)], \\ & [(n_{11}, n_{11}), (n_{20}, n_{20})], \\ & [(n_{11}, n_4), (n_{20}, n_9)], [(n_{20}, n_4)]\} \end{aligned}$$

Properties 2.5 and 2.6 of $A(k)$ equivalence can easily be bootstrapped to properties of $P(k)$ equivalence.

Proposition 2.10. *Let $D = (V, Ed, r, \lambda)$ be a document, let $k \in \mathbb{N}$ and let m_1, n_1, m_2, $n_2 \in V$ such that m_1 is an ancestor of n_1 and m_2 is an ancestor of n_2. If $(n_1, m_1) \equiv_{P(k+1)} (n_2, m_2)$ then $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$*

Proposition 2.11. *Let $D = (V, Ed, r, \lambda)$ be a document, and let $k \geq \mathbf{height}(D)$. Then $P(k)$ equivalence coincides with $P(\mathbf{height}(D))$ equivalence.*

As with $A(k)$ equivalence, we extend $P(k)$ as follows to capture upward paths to the root in a document.

Definition 2.12. *Let $D = (V, Ed, r, \lambda)$ be a document and let m_1, n_1, m_2, $n_2 \in V$ such that m_1 is an ancestor of n_1 and m_2 is an ancestor of n_2. Then, (n_1, m_1) and (n_2, m_2) are* $P(\infty)$ equivalent, *denoted $(n_1, m_1) \equiv_{P(\infty)} (n_2, m_2)$, if*

1. $\mathbf{distance}(n_1, m_1) = \mathbf{distance}(n_2, m_2)$*; and*
2. $n_1 \equiv_{A(\infty)} n_2$.

Proposition 2.8 for $A(\infty)$ equivalence can be bootstrapped to $P(\infty)$ equivalence in a straightforward manner.

Proposition 2.13. *Let $D = (V, Ed, r, \lambda)$ be a document. Then $P(\infty)$ equivalence coincides with $P(\mathbf{height}(D))$ equivalence.*

Note that $P(\infty)$ equivalence induces the 2-index proposed by Milo and Suciu [12].

As a closing remark, we observe from the definitions of $A(k)$ and $P(k)$ indices that, given an $A(k)$ index on a document D, it is straightforward to derive the corresponding $P(k)$ index for D, and vice versa.

3 Algebraic Characterizations of the $A(k)$ and $P(k)$ Indices

In this section, we give characterizations of the $A(k)$ and $P(k)$ family of indices based on notions of indistinguishability of nodes and pairs of nodes in fragments of the XPath-algebra. The notion of language indistinguishability on nodes is made precise in the following definition, following Gyssens et al. [6].

Definition 3.1. *Let $D = (V, Ed, r, \lambda)$ be a document, $m_1, m_2 \in V$, and $\mathscr{F}$ a fragment of the XPath-algebra. We say m_1 and m_2 are* indistinguishable by *$\mathscr{F}$, denoted $m_1 \equiv_{\mathscr{F}} m_2$, if for any expression E in $\mathscr{F}$, it is the case that $E(D)(m_1) = \emptyset$ if and only if $E(D)(m_2) = \emptyset$.*

Language indistinguishability on pairs of nodes is defined as follows.

Definition 3.2. *Let $D = (V, Ed, r, \lambda)$ be a document, $n_1, m_1, n_2, m_2 \in V$, and $\mathscr{F}$ a fragment of the XPath-algebra. We say (n_1, m_1) and (n_2, m_2) are* indistinguishable by *$\mathscr{F}$, denoted $(n_1, m_1) \equiv_{\mathscr{F}} (n_2, m_2)$, if for any expression E in $\mathscr{F}$, it is the case that $(n_1, m_1) \in E(D)$ if and only if $(n_2, m_2) \in E(D)$.*

Next, we introduce a family of algebras which we use to precisely characterize the $A(k)$ and $P(k)$ indices.

Definition 3.3. *We recursively define the* upward-k XPath algebras, *$\mathscr{U}(k)$ for each $k \in \mathbb{N}$, as follows.*

1. *$\mathscr{U}(0)$ = the set of XPath-algebra expressions without occurrences of the "$\downarrow$" and "$\uparrow$" operators.*
2. *For $k \geq 1$,*
   (a) *if $E \in \mathscr{U}(k-1)$, then $E \in \mathscr{U}(k)$;*
   (b) *$\uparrow \in \mathscr{U}(k)$;*
   (c) *if $E_1 \in \mathscr{U}(k)$ and $E_2 \in \mathscr{U}(k)$, then*
       - *$E_1 \star E_2 \in \mathscr{U}(k)$, for $\star = \cup, \cap, -$;*
   (d) *if $E_1 \in \mathscr{U}(k_1)$ and $E_2 \in \mathscr{U}(k_2)$, and $k_1 + k_2 \leq k$, then*
       - *$E_1 \diamond E_2 \in \mathscr{U}(k)$ and $E_1[E_2] \in \mathscr{U}(k)$; and*
3. *No other expressions are in $\mathscr{U}(k)$.*

As a very simple example of $\mathscr{U}(k)$ expressions, note that $\uparrow \diamond \uparrow \diamond \uparrow$ is in $\mathscr{U}(3)$ but not in $\mathscr{U}(2)$.

The following useful observation about the $\mathscr{U}(k)$ algebras can be shown by a simple inductive argument.

Proposition 3.4. *Let $D = (V, Ed, r, \lambda)$ be a document, $k \in \mathbb{N}$, $m, n \in V$, and $E \in \mathscr{U}(k)$. If $(n, m) \in E(D)$, then m is an ancestor of n such that $\mathbf{distance}(n, m) \leq k$.*

The $\mathscr{U}(k)$ algebras are extended as follows to an algebra with unrestricted use of the "$\uparrow$" primitive.

Definition 3.5. *The* upward XPath algebra *$\mathscr{U}(\infty)$ is the set of all XPath-algebra expressions without occurrences of the $\downarrow$ primitive.*

In other words, $\mathscr{U}(\infty) = \bigcup_{k \in \mathbb{N}} \mathscr{U}(k)$.

Compositions in which the number of nested "$\uparrow$" primitives is strictly greater than $\mathbf{height}(D)$ must return the empty set, and are therefore not very meaningful. The following result should therefore not come as a surprise.

Proposition 3.6. *The $\mathscr{U}(\infty)$ and $\mathscr{U}(\mathbf{height}(D))$ algebras are equivalent in expressive power.*

In particular, indistinguishability by $\mathscr{U}(\infty)$ and indistinguishability by $\mathscr{U}(\mathbf{height}(D))$ coincide.

3.1 Structural Characterizations of $\mathscr{U}(k)$ and $\mathscr{U}(\infty)$ Indistinguishability

In this section, we provide precise structural characterizations of indistinguishability in the upward algebras. In particular, for each $k \geqslant 0$, we show that on a fixed document, indistinguishability of nodes by the $\mathscr{U}(k)$ algebra corresponds precisely with $A(k)$ equivalence of nodes, and indistinguishability of node pairs by the $\mathscr{U}(k)$ algebra corresponds precisely with $P(k)$ equivalence. We also extend these results to the full upward algebra $\mathscr{U}(\infty)$. We follow the methodology of Gyssens et al. [6] to establish these correspondences.

The following facts are crucial in the sequel.

Lemma 3.7. *Let $D = (V, Ed, r, \lambda)$ be a document, $k \in \mathbb{N}$, and n_1, m_1, $n_2 \in V$ such that m_1 is an ancestor of n_1 and $\mathbf{distance}(n_1, m_1) \leq k$. If $n_1 \equiv_{A(k)} n_2$, then there exists $m_2 \in V$ such that m_2 is an ancestor of n_2 and $(n_1, m_1) \equiv_{P(\mathbf{distance}(n_1,m_1))} (n_2, m_2)$.[2] Furthermore, $m_1 \equiv_{A(k-\mathbf{distance}(n_1,m_1))} m_2$.*

Proof. By induction on k. For the base case, $k = 0$, clearly $m_1 = n_1$ and $\lambda(n_1) = \lambda(n_2)$. The statement holds for $m_2 = n_2$.

For $k \geq 1$, we can assume that the statement holds for $k-1$. If $n_1 \equiv_{A(k)} n_2$, then either (1) both n_1 and n_2 have no parents, or (2) they both have parents p_1 and p_2, resp., such that $p_1 \equiv_{A(k-1)} p_2$ (by definition of $A(k)$ equivalence). In case (1), clearly $m_1 = n_1$ and the statement holds for $m_2 = n_2$. In case (2), $\mathbf{distance}(p_1, m_1) \leq k-1$, and by the definition of $A(k)$ equivalence, $p_1 \equiv_{A(k-1)} p_2$. By the induction hypothesis, we have that there exists an ancestor m_2 of p_2 such that $(p_1, m_1) \equiv_{P(\mathbf{distance}(p_1,m_2))} (p_2, m_2)$ and $m_1 \equiv_{A(k-1-\mathbf{distance}(p_1,m_1))} m_2$. It readily follows that $(n_1, m_1) \equiv_{P(\mathbf{distance}(n_1,m_1))} (n_2, m_2)$ and $m_1 \equiv_{A(k-\mathbf{distance}(n_1,m_1))} m_2$. □

Proposition 3.8. *Let $D = (V, Ed, r, \lambda)$ be a document, $k \in \mathbb{N}$, $E \in \mathscr{U}(k)$, and $n_1, m_1, n_2, m_2 \in V$ such that m_1 is an ancestor of n_1 and m_2 is an ancestor of n_2, and $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$. If $(n_1, m_1) \in E(D)$, then $(n_2, m_2) \in E(D)$, and vice versa.*

Proof. First observe that it follows from $E \in \mathscr{U}(k)$ and $(n_1, m_1) \in E(D)$ that $\mathbf{distance}(n_1, m_1) \leq k$, by Proposition 3.4.

The proof is by induction on k. The base case, $k = 0$, follows straightforwardly from the definition of $P(0)$ equivalence and a simple structural induction on expressions in $\mathscr{U}(0)$. Now assume that $k \geq 1$, and that the statement holds for $0, 1, 2, \ldots, k-1$. The proof goes by structural induction on expressions in $\mathscr{U}(k)$. Thus, let $E \in \mathscr{U}(k)$.

- $E \in \mathscr{U}(k-1)$. The statement holds by the induction hypothesis.
- $E = \uparrow$. If $(n_1, m_1) \in \uparrow(D)$, then m_1 is the parent of n_1. Since $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$, it follows in particular that m_2 is the parent of n_2. We conclude that $(n_2, m_2) \in \uparrow(D)$.
- $E = E_1 \cup E_2$, for E_1 and $E_2 \in \mathscr{U}(k)$. Suppose $(n_1, m_1) \in E(D)$. Then $(n_1, m_1) \in E_1(D)$ or $(n_1, m_1) \in E_2(D)$. Without loss of generality, assume $(n_1, m_1) \in E_1(D)$. Then by structural induction, $(n_2, m_2) \in E_1(D)$, and we conclude $(n_2, m_2) \in E(D)$.
- $E = E_1 \cap E_2$ or $E = E_1 - E_2$, for E_1 and $E_2 \in \mathscr{U}(k)$. Similar to the previous case.
- $E = E_1 \diamond E_2$, for $E_1 \in \mathscr{U}(k_1)$ and $E_2 \in \mathscr{U}(k_2)$, such that $k_1 + k_2 \leq k$. Suppose $(n_1, m_1) \in E(D)$. Then there exists a node $w_1 \in V$ such that $(n_1, w_1) \in E_1(D)$ and $(w_1, m_1) \in E_2(D)$. By Lemma 3.4, $\mathbf{distance}(n_1, w_1) \leq k_1$ and $\mathbf{distance}(w_1, m_1) \leq k_2$. By Lemma 3.7, there exists a node $w_2 \in V$ such that $(n_1, w_1) \equiv_{P(\mathbf{distance}(n_1,w_1))} (n_2, w_2)$, and $w_1 \equiv_{A(k-\mathbf{distance}(n_1,w_1))} w_2$. Since, $k_2 \leq k - \mathbf{distance}(n_1, w_1)$, by Lemma 3.7, a node $m' \in V$ exists with $(w_1, m_1) \equiv_{P(\mathbf{distance}(w_1,m_1))} (w_2, m')$. Since $(n_1, w_1) \equiv_{P(\mathbf{distance}(n_1,w_1))} (n_2, w_2)$, and $(w_1, m_1) \equiv_{P(\mathbf{distance}(w_1,m_1))} (w_2, m')$, we know, by the definition of $\equiv_{P(k_1)}$ and $\equiv_{P(k_2)}$, that $\mathbf{distance}(n_2, w_2) = \mathbf{distance}(n_1, w_1)$ and $\mathbf{distance}(w_2, m') = \mathbf{distance}(w_1, m_1)$. Consequently, $\mathbf{distance}(n_2, m') = \mathbf{distance}(n_1, m_1)$, and since m' is the unique ancestor at this distance, we conclude that $m' = m_2$. Thus $(w_1, m_1) \equiv_{P(k_2)} (w_2, m_2)$. By the induction hypothesis, we can conclude that $(n_2, w_2) \in E_1(D)$ and $(w_2, m) \in E_2(D)$ and thus $(n_2, m_2) \in E(D)$.
- $E = E_1[E_2]$, for $E_1 \in \mathscr{U}(k_1)$ and $E_2 \in \mathscr{U}(k_2)$, such that $k_1 + k_2 \leq k$. Similar to the previous case. □

An important corollary of Proposition 3.8 is the following observation.

Proposition 3.9. *Let $E \in \mathscr{U}(k)$, $k \in \mathbb{N}$, and let $D = (V, Ed, r, \lambda)$ be a document. Then $E(D)$ is the union of some partition blocks of the $P(k)$-index.*

We are now prepared to establish the main results of this section.

Theorem 3.10. *Let $D = (V, Ed, r, \lambda)$ be a document, $n_1, n_2 \in V$, and $k \in \mathbb{N}$. Then $n_1 \equiv_{\mathscr{U}(k)} n_2$ if and only if $n_1 \equiv_{A(k)} n_2$.*

[2] And even stronger, $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$.

Proof. (If) Suppose $n_1 \equiv_{A(k)} n_2$ and let $E \in \mathscr{U}(k)$ such that $E(D)(n_1) \neq \emptyset$. Hence, there exists $m_1 \in V$ such that $(n_1, m_1) \in E(D)$. Then, by Lemma 3.7 and Proposition 3.8, there exists $m_2 \in V$ such that $(n_2, m_2) \in E(D)$, and therefore $E(D)(n_2) \neq \emptyset$. It follows symmetrically that if $E(D)(n_2) \neq \emptyset$, then $E(D)(n_1) \neq \emptyset$. We conclude that $n_1 \equiv_{\mathscr{U}(k)} n_2$.

(Only if) For the converse, assume that $n_1 \equiv_{\mathscr{U}(k)} n_2$. We first establish two facts.

1. $\lambda(n_1) = \lambda(n_2)$. Otherwise, consider the expression $\lambda(n_1) \in \mathscr{U}(k)$. Then $\lambda(n_1)(D)(n_1) \neq \emptyset$ and $\lambda(n_1)(D)(n_2) = \emptyset$, a contradiction.

2. if $k \geq 1$, then either n_1 and n_2 both have parents or are both the root. Otherwise, assume that, e.g., n_1 has a parent and n_2 is the root. Consider the expression $\uparrow$. Clearly, $\uparrow(D)(n_1) \neq \emptyset$, but $\uparrow(D)(n_2) = \emptyset$, a contradiction.

   We now prove the statement of the theorem by induction on k. For $k = 0$, this follows immediately from Fact 1.

   Therefore, consider the case $k \geq 1$, and assume the statement holds for $k - 1$. If n_1 and n_2 are both the root, then the statement holds trivially. In the remaining case, we know by Facts 1 and 2 that $\lambda(n_1) = \lambda(n_2)$, n_1 has a parent, say p_1, and n_2 has parent, say p_2. Assume that p_1 is not indistinguishable from p_2 by $\mathscr{U}(k-1)$. Then, there exists an expression E in $\mathscr{U}(k-1)$ such that $E(D)(p_1) \neq \emptyset$ and $E(D)(p_2) = \emptyset$, or vice-versa. Now consider the expression $F = \uparrow \diamond E$ which is in $\mathscr{U}(k)$. Clearly, $F(D)(n_1) \neq \emptyset$ and $F(D)(n_2) = \emptyset$, or vice-versa, a contradiction. Thus $p_1 \equiv_{\mathscr{U}(k-1)} p_2$, and therefore, by the induction hypothesis, $p_1 \equiv_{A(k-1)} p_2$, from which the statement of the theorem immediately follows.

□

Following a similar argument, we have the following characterization of $\mathscr{U}(k)$ indistinguishability on node pairs.

Theorem 3.11. *Let $D = (V, Ed, r, \lambda)$ be a document, $n_1, m_1, n_2, m_2 \in V$, and $k \in \mathbb{N}$. Then $(n_1, m_1) \equiv_{\mathscr{U}(k)} (n_2, m_2)$ if and only if $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$.*

We remind the reader that indistinguishability by $\mathscr{U}(\infty)$ and indistinguishability by $\mathscr{U}(\mathbf{height}(D))$ coincide (Proposition 3.6). Furthermore, by Proposition 2.8, $A(\infty)$ equivalence coincides with $A(\mathbf{height}(D))$ equivalence. Therefore, the result below immediately follows.

Theorem 3.12. *Let $D = (V, Ed, r, \lambda)$ be a document and $n_1, n_2 \in V$. Then $n_1 \equiv_{\mathscr{U}(\infty)} n_2$ if and only if $n_1 \equiv_{A(\infty)} n_2$.*

Similarly, by Proposition 2.13, $P(\infty)$ equivalence coincides with $P(\mathbf{height}(D))$ equivalence. Therefore, the result below immediately follows.

Theorem 3.13. *Let $D = (V, Ed, r, \lambda)$ be a document and n_1, m_1, n_1, $n_2 \in V$. Then $(n_1, m_1) \equiv_{\mathscr{U}(\infty)} (n_2, m_2)$ if and only if $(n_1, m_1) \equiv_{P(\infty)} (n_2, m_2)$.*

4 Labeling $A(k)$ and $P(k)$ Partitions with XPath Algebra Expressions

In the previous section we investigated the semantic relationships between $\mathscr{U}(k)$ and $\mathscr{U}(\infty)$ equivalence on the one hand, and the $A(k)$, $A(\infty)$, $P(k)$, and $P(\infty)$ indices on the other. There is also an alternative *syntactic* characterization of these relationships, which we develop in this section. First, a fundamental definition.

Definition 4.1. *Let $D = (V, Ed, r, \lambda)$ be a document, $k \in \mathbb{N}$, and $n \in V$. Let the k-*ancestor node path *of n be the unique list of nodes $n_0, \ldots, n_\ell$ on the path from $n = n_0$ up towards the root node r, of length $\ell = \min\{k, \mathbf{distance}(n, r)\}$. Note that when $k \geqslant \mathbf{distance}(n, r)$, it is the case that $n_\ell = r$. For $i \geqslant 0$, the i^{th} k-*ancestor path expression *of n is the $\mathscr{U}(\max\{i, \ell\})$ expression $AP^i_{k,n}$ defined in Figure 3.*[3]

The following two observations follow directly from the definitions of $A(k)$ and $P(k)$ equivalence and the definition of i^{th} k-ancestor path expressions.

Proposition 4.2. *Let $D = (V, Ed, r, \lambda)$ be a document, $n_1, n_2 \in V$, and $k \in \mathbb{N}$. Then the following are equivalent:*

(1) $n_1 \equiv_{A(k)} n_2$
(2) $AP^0_{k,n_1} = AP^0_{k,n_2}$

Proposition 4.3. *Let $D = (V, Ed, r, \lambda)$ be a document, $n_1, n_2, m_1, m_2 \in V$, and $k \in \mathbb{N}$, such that m_1 is an ancestor of n_1 and m_2 is an ancestor of n_2. Then the following are equivalent:*

(1) $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$
(2) $AP^{\mathbf{distance}(n_1,m_1)}_{k,n_1} = AP^{\mathbf{distance}(n_2,m_2)}_{k,n_2}$

We next show that these statements can be strengthened. For this, we need two tools. To compare i^{th} k-ancestor path expressions, we introduce the following definition.

Definition 4.4. *Let $D = (V, Ed, r, \lambda)$ be a document, $i, k \in \mathbb{N}$, and $m, n \in V$. Suppose the k-ancestor node paths of m and n are $m_0, \ldots, m_{\ell_m}$ and $n_0, \ldots, n_{\ell_n}$, respectively. If $\ell_m < \ell_n$ and $\lambda(m_j) = \lambda(n_j)$ for all $0 \leqslant j \leqslant \ell_m$, we say that $AP^i_{k,m}$ is a* prefix *of $AP^i_{k,n}$, denoted $AP^i_{k,m} \prec AP^i_{k,n}$.*

[3] Where $\uparrow^0 = \varepsilon$ and for $i > 0$, $\uparrow^i = \underbrace{\uparrow \diamond \cdots \diamond \uparrow}_{i \text{ times}}$.

$$AP^i_{k,n} = \begin{cases} \lambda(n_0)\diamond\uparrow\diamond\cdots\diamond\uparrow\diamond\lambda(n_i)[\uparrow\diamond\lambda(n_{i+1})\ \diamond\cdots\diamond\uparrow\diamond\lambda(n_\ell)] & \text{if } i < \ell \\ \lambda(n_0)\diamond\uparrow\diamond\cdots\diamond\uparrow\diamond\lambda(n_\ell)\diamond\uparrow^{i-\ell} & \text{if } i \geqslant \ell \end{cases}$$

Figure 3. The i^{th} k-ancestor path expression of node n having k-ancestor node path $(n_0, \ldots, n_\ell)$.

To single out partition blocks of the $A(k)$ and $P(k)$ indices, we introduce the following class of expressions derived from the $AP^i_{k,n}$ expressions above.

Definition 4.5. *Let $D = (V, Ed, r, \lambda)$ be a document, $k \in \mathbb{N}$, and $n, m \in V$ such that m is the i^{th} ancestor of n, for some $i \geqslant 0$. Then the k*-partition labeling expression *for (n, m) is the $\mathscr{U}(\mathbf{height}(D))$ expression*

$$PL_{k,(n,m)} = AP^i_{k,n} - \bigcup_{\substack{n' \in V \\ \& \\ AP^i_{k,n} \prec AP^i_{k,n'}}} AP^i_{k,n'}$$

We are now prepared to strengthen Propositions 4.2 and 4.3.

Theorem 4.6. *Let $D = (V, Ed, r, \lambda)$ be a document, $n_1, n_2 \in V$, and $k \in \mathbb{N}$. Then the following are equivalent:*

(1) $n_1 \equiv_{A(k)} n_2$
(2) $(n_2, n_2) \in PL_{k,(n_1,n_1)}(D)$
(3) $PL_{k,(n_1,n_1)} = PL_{k,(n_2,n_2)}$

Theorem 4.7. *Let $D = (V, Ed, r, \lambda)$ be a document, $n_1, n_2, m_1, m_2 \in V$, and $k \in \mathbb{N}$, such that m_1 is an ancestor of n_1 and m_2 is an ancestor of n_2. Then the following are equivalent:*

(1) $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$
(2) $(n_2, m_2) \in PL_{k,(n_1,m_1)}(D)$
(3) $PL_{k,(n_1,m_1)} = PL_{k,(n_2,m_2)}$

We give a proof of Theorem 4.7; the proof of Theorem 4.6 is just a special case of this proof.

Proof. ($1 \Rightarrow 2$) If $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$, then $n_1 \equiv_{A(k)} n_2$ and $\mathbf{distance}(n_1, m_1) = \mathbf{distance}(n_2, m_2)$. Then by construction of the $\mathbf{distance}(n_1, m_1)$ k-ancestor path expression $AP^{\mathbf{distance}(n_1,m_1)}_{k,n_1}$, it must be the case that $(n_2, m_2) \in AP^{\mathbf{distance}(n_1,m_1)}_{k,n_1}(D)$. Furthermore, since $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$, it must be the case that $(n_2, m_2) \notin AP^{\mathbf{distance}(n_1,m_1)}_{k,n'}(D)$ for any $n' \in V$ such that $AP^{\mathbf{distance}(n_1,m_1)}_{k,n_1}$ is a prefix of $AP^{\mathbf{distance}(n_1,m_1)}_{k,n'}$. We conclude that $(n_2, m_2) \in PL_{k,(n_1,m_1)}(D)$, by definition of k-partition labeling expressions.

($2 \Rightarrow 1$) If $(n_2, m_2) \in PL_{k,(n_1,m_1)}(D)$ then (1) $(n_2, m_2) \in AP^{\mathbf{distance}(n_1,m_1)}_{k,n_1}(D)$, and (2) $(n_2, m_2) \notin AP^{\mathbf{distance}(n_1,m_1)}_{k,n'}(D)$ for any $n' \in V$ such that $AP^{\mathbf{distance}(n_1,m_1)}_{k,n_1}$ is a prefix of $AP^{\mathbf{distance}(n_1,m_1)}_{k,n'}$. From (1) we conclude that $\mathbf{distance}(n_1, m_1) = \mathbf{distance}(n_2, m_2)$ and from (1) and (2) combined that $n_1 \equiv_{A(k)} n_2$. Hence, $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$.

($1 \Leftrightarrow 3$) This follows directly from Proposition 4.3 and the definition of k-partition labeling expressions. □

Remark 4.8. *In summary, Theorems 4.6 and 4.7 show that each $A(k)$ and $P(k)$ partition block of a document D can be assigned an expression $E \in \mathscr{U}(\mathbf{height}(D))$ such that $E(D)$ yields precisely those members of the block.*

With these observations in hand, we close the section with a *conservative extension* result for $\mathscr{U}(\infty)$.

Theorem 4.9. *Let $k \in \mathbb{N}$. For each expression $E \in \mathscr{U}(\infty) - \mathscr{U}(k)$, there exists an expression $E_k \in \mathscr{U}(k)$ such that for all documents D with $\mathbf{height}(D) = k$, $E(D) = E_k(D)$.*

Proof. Assume for sake of contradiction that the statement does not hold, i.e., there exists an expression $E \in \mathscr{U}(\infty) - \mathscr{U}(k)$, such that for every expression $E_k \in \mathscr{U}(k)$ there exists a document D with $\mathbf{height}(D) = k$ where $E(D) \neq E_k(D)$.

For any document D with $\mathbf{height}(D) = k$, we know from Proposition 2.13 that $P(k)$ equivalence coincides with $P(\infty)$ equivalence. Furthermore, we know from Theorem 3.13 that $P(\infty)$ equivalence coincides with $\mathscr{U}(\infty)$ equivalence. Hence, it follows that E evaluated on D is equal to the union of some of the partition blocks of the $P(k)$ index for D. In this case, we observe from Remark 4.8 that E can be rewritten as a union of $\mathscr{U}(k)$ partition labeling expressions. In other words, there exists an expression $E_k \in \mathscr{U}(k)$ such that $E(D) = E_k(D)$, a contradiction of our assumption. Hence, we conclude that the statement holds. □

5 XPath Query Evaluation with $A(k)$ and $P(k)$ Indices

In this section, we consider strategies to evaluate XPath expressions, leveraging the results and characterizations of the $A(k)$ and $P(k)$ indices (Sections 3 and 4). First, we discuss how $A(k)$ and $P(k)$ indices can be used in the evaluation of upward XPath expressions, as expressed in the

$\mathscr{U}(\infty)$ and $\mathscr{U}(k)$ algebras. Then, we focus on the evaluation of XPath algebra query expressions that can be expressed in certain **downward** XPath algebras – namely, the *downward algebra* $\mathscr{D}(\infty)$ and the *downward-k-algebras* $\mathscr{D}(k)$ (for each $k \in \mathbb{N}$). These algebras are defined just like the upward algebras except that one is restricted to using $\downarrow$ operations instead of $\uparrow$ operations.

5.1 Evaluating Upward Expressions

Our analysis of the $A(k)$ and $P(k)$ indices shows that they are the ideal index structures for evaluation of $\mathscr{U}(k)$ expressions. Indeed, consider an expression $E \in \mathscr{U}(k)$, a document D, and its $P(k)$-index. Proposition 3.9 states that the evaluation of E on D, $E(D)$, can be evaluated as the union of certain $P(k)$ partition blocks. Now, since $A(k)$ and $P(k)$ indices are inter-derivable, it is clear that an $A(k)$-index on D can also be used effectively in the evaluation of $E(D)$.

Actually, any $\mathscr{U}(\infty)$ expression can also make effective use of an existing $A(k)$ or $P(k)$ index. For example, consider the expression $E = \uparrow \diamond \uparrow \diamond \uparrow \diamond \uparrow$ and suppose that we have only the $P(2)$-index available. Since $E \in \mathscr{U}(4)$, in general $E(D)$ will not neccesarily be a union of some of the partition blocks of $P(2)$. However, if we *decompose* E as $E_1 \diamond E_2$, where $E_1 = \uparrow \diamond \uparrow$ and $E_2 = \uparrow \diamond \uparrow$, then $E(D) = E_1(D) \bowtie E_2(D)$, and since E_1 and E_2 are in $\mathscr{U}(2)$, they can obviously be evaluated directly with the existing index as discussed above.

5.2 Evaluating Downward Expressions

As shown in XPath use cases and various XML benchmarks, over ninety percent of XPath queries used in real world applications use navigation just along the parent-child axis, rather than mixing the parent-child and child-parent axes. These queries can be naturally expressed in the $\mathscr{D}(\infty)$ and $\mathscr{D}(k)$ algebras.

A natural way to evaluate $\mathscr{D}(\infty)$ expressions is to perform **top-down** navigation. We demonstrate that such expressions can also be evaluated using path decomposition and $A(k)$ and $P(k)$ indices.

The main idea behind turning top-down evaluation into bottom-up evaluation is to convert a downward expression, or certain of its sub-expressions, into an upward expression using a technique that we will refer to as "inverting expressions." We first illustrate this technique on predicate-free downward expressions, and then consider general downward expressions.

5.2.1 Downward Expressions without Predicates

In general, predicate-free expressions in the downward algebras can be "inverted" into predication-free expressions in the corresponding upward algebras using the rewrite rules shown in Table 3.

E	$\rightarrow$	E^{-1}
ϵ	$\rightarrow$	ϵ
$\emptyset$	$\rightarrow$	$\emptyset$
$\downarrow$	$\rightarrow$	$\uparrow$
$\hat{\lambda}$	$\rightarrow$	$\hat{\lambda}$
$E_1 \cup E_2$	$\rightarrow$	$E_1^{-1} \cup E_2^{-1}$
$E_1 \cap E_2$	$\rightarrow$	$E_1^{-1} \cap E_2^{-1}$
$E_1 - E_2$	$\rightarrow$	$E_1^{-1} - E_2^{-1}$
$E_1 \diamond E_2$	$\rightarrow$	$E_2^{-1} \diamond E_1^{-1}$.

Table 3. Inversion Rewrite Rules for $\mathscr{D}(\infty)$.

As discussed above, an upward algebraic expression in $\mathscr{U}(k)$ can be evaluated by $P(k)$-index lookup and set unions. Therefore, a predicate-free downward algebraic expression in $\mathscr{D}(k)$ can be evaluated by inversion, $P(k)$-index lookup, and set unions.

5.2.2 Downward Expressions with Predicates

Now consider the evaluation of downward algebra expressions wherein predicate operations occur. A simple example is the expression $E_2 = \downarrow [\downarrow]$. Applied to a document, E_2 evaluates to the document's parent-child pairs for children that have at least one child itself. As in Section 5.2.1, to evaluate E_2 on a document D, we could consider the concept of inverting E_2 into an expression $E_2^{-1} \in \mathscr{U}(\infty)$ such that $E_2(D) = (E_2^{-1}(D))^{-1}$. Unfortunately, this approach does work here. In fact, we can construct a document D_2 such for **each** expression $F \in \mathscr{U}(\infty)$, $E_2(D_2) \neq F(D)^{-1}$.

Clearly E_2 is equivalent to the XPath-algebra expression $\downarrow \diamond \downarrow \diamond \uparrow$. Notice that this expression is neither a downward nor an upward expression. However its sub-expression $G_1 = \downarrow \diamond \downarrow$ is in $\mathscr{D}(2)$ and its sub-expression $G_2 = \uparrow$ is in $\mathscr{U}(1)$. Applying the inversion technique described in Section 5.2.1 to G_1, the evaluation $E_2(D)$ on a document D, can be accomplished by computing the relation $(G_1^{-1}(D))^{-1} \bowtie G_2(D)$. And, as indicated in this section, the evaluations of $G_1^{-1}(D) = \uparrow \diamond \uparrow (D)$ and $G_2(D) = \uparrow (D)$ can be done efficiently using the $P(2)$ and $P(1)$ indices of D, respectively.

Given that the selectivity of a longer path is no larger than that of short sub-paths of the path, evaluating G_1 reduces the search space to the minimum that can be obtained on such a chain expression. Starting from any given node, upward navigation in an XML data tree, unlike downward ones, has one and only one route to follow, which is to reach its parent. Therefore, it is reasonable to claim that the result of $G_1^{-1}(D)$ is substantially smaller than that of $G_2(D)$, and the $\bowtie$ operation can be further optimized by $G_1^{-1}(D)$ followed by an upward navigation.

We will now consider a slightly more complicated downward expression $E_3 \in \mathscr{D}(3)$ which retrieves information about leaders of projects that have a web site:

$$E_3 = \texttt{Department} \diamond \downarrow \diamond \texttt{Project}[\downarrow \diamond \texttt{Web}] \diamond \downarrow \diamond \texttt{Lead}.$$

E_3 can be represented as an expression pattern tree, as illustrated in Figure 4(a). The shaded node can be interpreted as the "answer" of E_3.

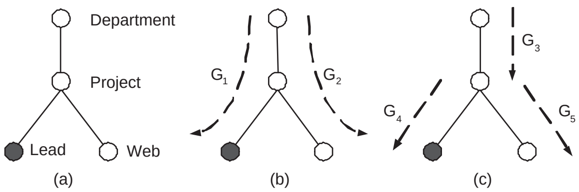


Figure 4. Chain pattern tree for E_3.

First, assume there exists a $P(2)$-index (i.e., where $k = 2$, the height of the expression). To take full advantage of the $P(2)$-index is to find the longest sub-expressions that can be inverted and evaluated directly on it. As shown in Figure 4(b), there are two natural chains of length 2, G_1 and G_2, in the pattern tree of E_3. There are also natural chains of length 1, G_3, G_4, and G_5, as shown in Figure 4(c).

Using G_1, G_2, and G_4, the expression E_3 is equivalent to the expression H_1 defined as follows:

$$H_1 = ((G_1 \diamond \uparrow) \cap (G_2 \diamond \uparrow)) \diamond G_4,$$

and therefore, for a document D, $E_3(D)$ can be computed as follows:

$$E_3(D) = \begin{pmatrix} ((G_1^{-1}(D))^{-1} \bowtie \uparrow (D)) \\ \cap \\ ((G_2^{-1}(D))^{-1} \bowtie \uparrow (D)) \end{pmatrix} \bowtie (G_4^{-1}(D))^{-1}.$$

All sub-expressions in this transformed expression of E_3 are in $\mathscr{U}(2)$, and hence can be be directly evaluated using the $P(2)$ partition blocks without any navigation.

However, it may not always be the case that $P(k)$-partitions exist for k larger than or equal to the height of the query expression. Again, for E_3, consider the case when only a $P(1)$-index is available. In this case, the longest path expressions that can take advantage of the index are of length 1. Such expressions are G_3, G_4 and G_5. Using these subexpressions, E_3 is equivalent with the expression H_2 defined as follows:

$$H_2 = (((G_3 \diamond G_4) \diamond \uparrow) \cap ((G_3 \diamond G_5) \diamond \uparrow)) \diamond G_4.$$

Similar to the generalization from E_2 to E_3, the *decomposition + inversion* techniques can be used to transform any arbitrary expression in the downward algebra $\mathscr{D}(\infty)$ to a set of predicate-free sub-expressions in $\mathscr{D}(\infty)$ that can be inverted to corresponding expressions in the upward algebra $\mathscr{U}(\infty)$, which in turn can be evaluated directly by $P(k)$-index lookups and set unions. The decomposition is partially determined by k, the degree of local similarity in the $P(k)$-indices which are available. Furthermore, this algebraic transformation only provides guidelines for evaluating each such expression. The optimal physical plan to evaluate an XPath query depends on other factors, such as the cardinality of the intermediate results of each sub-expression, and the cost model of the join operations.

6 Future Directions

It is clear that many aspects of the query evaluation guidelines presented in the previous section warrant further investigation. We close by listing a few of the most interesting directions for research. First, a more thorough study of query decomposition and inversion is crucial. Second, it is interesting to develop approaches to quickly identify which partition blocks correspond to an arbitrary given input expression, using the labeling techniques of Section 4. Finally, the results of this paper are applicable in workload driven scenarios [7, 13, 15], such as determining which indices to materialize and which to derive, as a function of the characteristics of an observed workload.

Acknowledgments. We thank Changqing Lin for discussions on the $A(k)$ indices.

References

[1] T. Bray, J. Paoli, C. Sperberg-McQueen, and F. Yergeau. Etensible Markup Language (XML) 1.0 (third edition) - W3C recommendation, 2004.

[2] N. Bruno et al. Holistic twig joins: optimal XML pattern matching. In *SIGMOD*, 2002.

[3] D. Chamberlin et al. XQuery 1.0: An XML query language, W3C, 2003.

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

[5] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB*, pages 436–445, 1997.

[6] 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 *ACM PODS*, pages 318–327, 2006.

[7] H. He and J. Yang. Multiresolution indexing of XML for frequent queries. In *IEEE ICDE*, 2004.

[8] R. Kaushik, P. Bohannon, J. F. Naughton, and H. F. Korth. Covering indexes for branching path queries. In *SIGMOD*, 2002.

[9] R. Kaushik, R. Krishnamurthy, J. F. Naughton, and R. Ramakrishnan. On the integration of structure indexes and inverted lists. In *ACM SIGMOD*, 2004.

[10] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting local similarity for efficient indexing of paths in graph structured data. In *IEEE ICDE*, 2002.

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

[12] T. Milo and D. Suciu. Index structures for path expressions. In *ICDT*, pages 277–295, 1999.

[13] J.-K. Min, C.-W. Chung, and K. Shim. An Adaptive Path Index for XML Data using the Query Workload. *Inf. Systems*, 30(6):467–487, 2005.

[14] M. Moro et al. Tree-pattern queries on a lightweight XML processor. In *VLDB*, 2005.

[15] C. Qun et al. D(k)-index: An adaptive structural summary for graph-structured data. In *SIGMOD*, 2003.

[16] P. Ramanan. Covering indexes for XML queries: Bisimulation - simulation = negation. In *VLDB*, 2003.

[17] K. Runapongsa, J. M. Patel, R. Bordawekar, and S. Padmanabhan. XIST: An XML index selection tool. In *XSym*, pages 219–234, 2004.

[18] Y. Wu et al. Structural joins: A primitive for efficient XML query pattern matching. In *ICDE*, 2002.

[19] K. Yi, H. He, I. Stanoi, and J. Yang. Incremental maintenence of XML structural indexes. In *ACM SIGMOD*, pages 491–502, 2004.

[20] C. Zhang et al. On supporting containment queries in relational database management systems. In *SIGMOD*, 2001.