A Methodology for Coupling Fragments of XPath with Structural Indexes for XML Documents ⋆

George H.L. Fletcher [a], Dirk Van Gucht [b], Yuqing Wu [b], Marc Gyssens [c], Sofía Brenes [b], Jan Paredaens [d]

[a] *Washington State University, Vancouver*

[b] *Indiana University, Bloomington*

[c] *Hasselt University & Transnational University of Limburg*

[d] *University of Antwerp*

Abstract

We introduce a new methodology for coupling *language*-induced partitions and *index*-induced partitions on XML documents that is aimed for the benefit of efficient evaluation of XPath queries. In particular, we identify XPath fragments which are ideally coupled with the newly introduced $P(k)$-partition which has its definition grounded in the well-known $A(k)$ structural index and its associated partition. We then utilize these couplings to investigate fundamental questions about the use of structural indexes in XPath query evaluation.

Key words: XML, XPath, Structural Indexes, Query Processing

⋆ This paper is a revised and extended version of the paper "A Methodology for Coupling Fragments of XPath with Structural Indexes for XML Documents" presented at DBPL 2007, the 11th International Symposium on Database Programming Languages, Vienna, Austria, 2007.

Email addresses: fletcher@vancouver.wsu.edu (George H.L. Fletcher), vgucht@cs.indiana.edu (Dirk Van Gucht), yuqwu@indiana.edu (Yuqing Wu), marc.gyssens@uhasselt.be (Marc Gyssens), sbrenesb@cs.indiana.edu (Sofía Brenes), jan.paredaens@ua.ac.be (Jan Paredaens).

1 Introduction

Supporting efficient access to XML data using XPath [4] continues to be an important research problem [7,8,14]. XPath queries are used to specify node-labeled trees which match portions of the hierarchical XML data. In XPath query evaluation, indexes similar to those used in relational database systems—namely, value indexes on tags and text values—are first used, together with structural join algorithms [1,3,8,23]. This approach turns out to be simple and efficient. However, the structural containment relationships native to XML data are not directly captured by value indexes.

To directly capture the structural information of XML data, a family of structural indexes has been introduced. DataGuide [6] was the first to be proposed, followed by the 1-index [15], which is based on the notion of bisimulation [21] among nodes in an XML document. These indexes can be used to evaluate some path expressions accurately without accessing the original data graph. Milo and Suciu [15] also introduced the 2-index and T-index, based on similarity of pairs (vectors) of nodes. Unfortunately, these and other early structural indexes tend to be too large for practical use because they typically maintain too fine-grained structural information about the document [11,19].

To remedy this, Kaushik et al. introduced the $A(k)$-index which uses a notion of bisimilarity on nodes relativized to paths of length k [13]. 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 $A(k)$ indexes. The $D(k)$-index [18] and $M(k)$-index [10] extend the $A(k)$-index to adapt to query workload. Yi et al. [22] developed update techniques for the $A(k)$-index and 1-index. Finally, the integrated use of structural and value indexes has been explored [12], and there have also been investigations on covering indexes [11,19] and index selection [17,20].

The introduction of structural indexes for XML data has lead to significant improvements in the performance of XPath query evaluation. As was demonstrated empirically, the performance benefits of these indexes are most dramatic when queries "match" the index definitions [13]. To date, however, there lacks a formal understanding of this notion of queries matching indexes. This leads to some fundamental questions about using structural indexes in query evaluation:

(1) For which fragments of XPath are particular structural indexes *ideally* suited?
(2) For these fragments, how are its expressions efficiently 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 present a methodology for investigating such questions and apply it to the important special case of the $A(k)$-indexes. For question (1), we begin by noting that the $A(k)$-index of a document induces a partitioning on its nodes. Recently, an approach has been proposed for considering partitioning XML documents based on notions of query indistinguishability of nodes and paths, relative to particular fragments of XPath [9]. If we apply this approach to show that there exists a fragment of XPath which induces a partition identical to the $A(k)$-partition, then we can speak of an "ideal" match between the index and this fragment. Given this ideal coupling, we can then turn to a principled investigation of questions (2) and (3). A main contribution of this paper is the identification of such a fragment of XPath.

Before going into the technical details of the various steps we take in our methodology, we illustrate the general approach with a simple example coming from relational databases. Note that the results in this example are well-known, and as such do not add to the results of this paper.

1.1 A motivating example

Consider the B$^+$-tree index on a column A of a relation R [5]. Clearly, this index induces a partition on the tuples of R: tuples t_1 and t_2 in R will be in the same partition block[1] if and only if $t_1(A) = t_2(A)$. We will call this partition the *B^+-tree-partition* on column A of R, and denote it as $\mathsf{Btree}(A, R)$. (For emphasis, observe that a B$^+$-tree *index* on A of R is different than the $\mathsf{Btree}(A, R)$*-partition*. The first is a tree data structure, whereas the second is a partition on R.)

Next, consider the relational algebra, and in particular its sub-algebra consisting of the *range queries*. In this example, we focus on such queries as they are specified on attribute A of R. We will denote this class of queries by $\mathsf{RangeQ}(A, R)$. Its queries are of the form

$$\sigma_{((a_1 \leq A \leq a_2) \texttt{ or } \cdots \texttt{ or } (a_{2n-1} \leq A \leq a_{2n}))}(R).^{2}$$

The $\mathsf{RangeQ}(A, R)$ algebra defines a partition on R, called the $\mathsf{RangeQ}(A, R)$-*partition* of R, and is defined as follows: tuples t_1 and t_2 in R are placed in the

[1] "Block" stands for an element of a partition, not to be confused with a block on a disk.

[2] For simplicity, we will assume that all the a_i values occur in the A-column of R.

same block of the $\mathsf{RangeQ}(A, R)$-partition if for *any* query Q in $\mathsf{RangeQ}(A, R)$, $t_1 \in Q(R)$ if and only if $t_2 \in Q(R)$. In other words, t_1 and t_2 can not be distinguished by any query in $\mathsf{RangeQ}(A, R)$, i.e., either t_1 and t_2 are both in $Q(R)$, or they are both not in $Q(R)$. An important property of the $\mathsf{RangeQ}(A, R)$-partition is that for each query $Q \in \mathsf{RangeQ}(A, R)$, there exists a subset of blocks in the partition such that $Q(R)$ is the union of these blocks.

A natural question that arises now is to ask if the Btree-partition and the RangeQ-partition are the same. It should come as no surprise that this is indeed the case. This is captured in the following fact.

Proposition 1 [Btree-RangeQ **Coupling Theorem**]

Let R be a relation and let A be one of its attributes. The $\mathsf{Btree}(A, R)$-partition and the $\mathsf{RangeQ}(A, R)$-partition are the same.

PROOF. We give a proof of this statement, not because it is difficult, but because its structure reveals the strategy that we will follow to prove an analogous theorem for the XML case (Theorem 18).

(1) Let tuples t_1 and t_2 be in the same block of the $\mathsf{Btree}(A, R)$-partition. Then, by definition, $t_1(A) = t_2(A)$. Consider now an arbitrary range query Q. Then clearly, if $t_1(A)$ (and therefore also $t_2(A)$) is in the range of Q then t_1 and t_2 are both in $Q(R)$, but if $t_1(A)$ is not in the range of Q, then they are both not in $Q(R)$. Consequently, t_1 and t_2 are in the same block of the $\mathsf{RangeQ}(A, R)$-partition.

(2) Let tuples t_1 and t_2 be in different blocks of the $\mathsf{Btree}(A, R)$-partition. Then, by definition, $t_1(A) \neq t_2(A)$. Let $a = t_1(A)$. Then the range query $\mathtt{label}_a := \sigma_{A=a}(R)$ has t_1 in its result, but not t_2. Thus t_1 and t_2 are in different blocks of the $\mathsf{RangeQ}(A, R)$-partition, and the proof is done.

An immediate consequence of Proposition 1 is that each range query evaluated on R is equal to the union of a family of blocks of the $\mathsf{Btree}(A, R)$-partition.

Proposition 2 [Btree-RangeQ **Block-Union Theorem**] *Let R be a relation, let A be one of its attributes, and let $Q \in \mathsf{RangeQ}(A, R)$. Then there exists a class $\mathcal{B}_Q$ of partition blocks of the $\mathsf{Btree}(A, R)$-partition such that*

$$Q(R) = \bigcup_{B \in \mathcal{B}_Q} B.$$

Note that the Btree-RangeQ Block-Union Theorem can provide guidance and insight in the processing of queries in richer relational fragments.

In the second part of the proof of Proposition 1, observe that the range query $\texttt{label}_a$ has the property that it uniquely identifies the block of the $\mathsf{RangeQ}(A, R)$-partition consisting of the tuples of R that are indistinguishable from t_1 by any query in $\mathsf{RangeQ}(A, R)$. We will call $\texttt{label}_a$ a *labeling query* and its defining a-value a *label*. Now as a consequence of Proposition 2 we have that evaluating a range query $Q \in \mathsf{RangeQ}(A, R)$ can be done by forming a union of such labeling expressions applied to R.

Proposition 3 [Btree-RangeQ **Label-Union Theorem**]

Let R be a relation and A one of its attributes. Then for each query $Q \in \mathsf{RangeQ}(A, R)$, there is a set of labeling queries $\mathcal{L}_Q \subseteq \mathsf{RangeQ}(A, R)$ such that

$$Q(R) = \bigcup_{\texttt{label} \in \mathcal{L}_Q} \texttt{label}(R).$$

Obviously, in practice we do not want to evaluate the labeling queries $\texttt{label} \in \mathcal{L}_Q$ directly on R, but rather we would want a data structure that stores each result $\texttt{label}(R)$. If such a data structure supports efficient look-up of the tuples in the partition block associated with each labeling expression $\texttt{label}$, then evaluation of Q can be done by simply streaming out these tuples. Of course, such a data structure is the B^+-tree index. So, in a formal sense we have shown that range queries match ideally with B^+-tree indexes, which of course is a well-known fact.

1.2 Paper overview

We proceed as in this motivating example, for structural indexes and the XPath query language. Specifically, we have the following:

- We introduce the family of $P(k)$-partitions, which are derivatives of the family of $A(k)$-partitions. It turns out that this new class of partitions is fundamental for establishing the results which follow.
- We then introduce a family of upward XPath algebras, $\mathscr{U}(k)$, and show that the $P(k)$-partition and the partition induced by the $\mathscr{U}(k)$ algebra are the same. As a consequence of this, we have that the evaluation of a $\mathscr{U}(k)$ query is equal to the union of some blocks of the $P(k)$-partition.
- Based on this result, we then develop guidelines for the use of a $P(k)$-partition in the evaluation of general XPath queries.
- Following this, we show that for each block in the $P(k)$-partition a labeling expression in $\mathscr{U}(k)$ can be constructed which uniquely identifies the block. Thus, we conclude that each query in $\mathscr{U}(k)$ can be rewritten as the union of some $\mathscr{U}(k)$ block labeling expressions.

These results indicate research directions into new data structures to support efficient evaluation of general XPath queries.

2 Coupling indexes and XPath fragments

In this section, we set out to apply the methodology described in the motivating relational example to the XML case.

2.1 The XML data model

We begin by introducing the document data model that will be used in this paper. Our data model is a simplified version of the XML data model wherein we view a document as a labeled tree.

Definition 4 *A* document D *is a 4-tuple* (V, Ed, r, λ)*, with* V *the finite set of nodes,* $Ed \subseteq V \times V$ *a tree of parent-child edges,* $r \in V$ *the root, and* $\lambda\colon V \to \mathcal{L}$ *a node-labeling function into a countably infinite set of labels* $\mathcal{L}$*.*

Given a document, it is useful to introduce the concept of its paths. We define the set of *paths* of a document D, denoted $\mathsf{Paths}(D)$, as the set $V \times V$. So, for us a path is not a sequence of nodes, but rather a pair. This makes sense however, since a pair of nodes $(n, m) \in \mathsf{Paths}(D)$ identifies the unique (shortest) path from node n to node m in D. The set of *downward-paths*, $\mathsf{DownPaths}(D)$, consists of the paths (n, m) where n is an ancestor of m. Similarly, the set of *upwards-paths*, $\mathsf{UpPaths}(D)$, consists of the paths (n, m) where n is a descendant of m. Furthermore, for $k \in \mathbb{N}$, $\mathsf{DownPaths}(D, k)$ ($\mathsf{UpPaths}(D, k)$) are those paths in $\mathsf{DownPaths}(D)$ (in $\mathsf{UpPaths}(D)$, respectively) of length at most k. For example, in document D of Figure 1, the path (n_1, n_1) is a member of both $\mathsf{DownPaths}(D, 0)$ and $\mathsf{UpPaths}(D, 0)$, the paths (n_1, n_1), (n_1, n_4), and (n_1, n_9) are in $\mathsf{DownPaths}(D, 2)$, and their corresponding inverse paths (n_1, n_1), (n_4, n_1), and (n_9, n_1) are in $\mathsf{UpPaths}(D, 2)$. The paths (n_9, n_{12}) and (n_1, n_{19}) are in neither $\mathsf{DownPaths}(D, 2)$ nor $\mathsf{UpPaths}(D, 2)$.

2.2 The $A(k)$-partition of a document

Given a labeled semi-structured document [3] and a natural number k, Kaushik et al. [13] introduced the $A(k)$-*index* for this document.

[3] A semi-structured document does not have to be a tree. In particular, it is possible that a node has multiple parents.

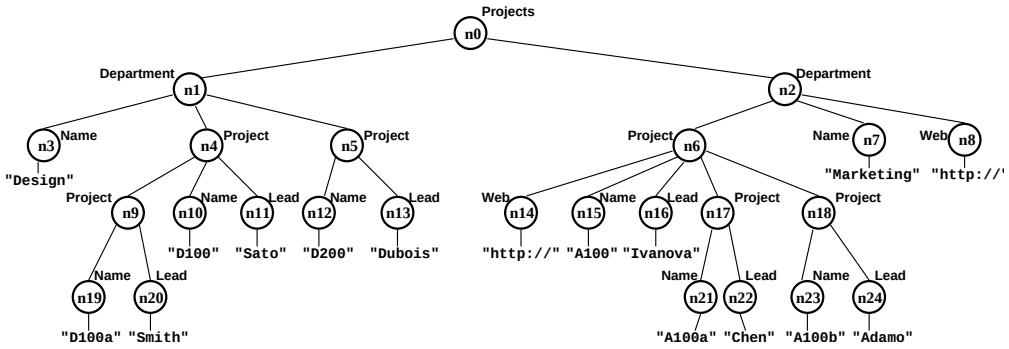


Fig. 1. An XML document. For reference, non-leaf nodes are given unique IDs.

The index is built on the partition induced by a certain bisimilarity equivalence relation on the nodes in the document. When specialized to a document, as defined here, the definition of this bisimilarity equivalence is as follows.

Definition 5 *Let $D = (V, Ed, r, \lambda)$ be a document, $n_1, n_2 \in V$, and let $k \in \mathbb{N}$. We say that n_1 and n_2 are* $A(k)$-equivalent in D, *denoted $n_1 \equiv_{A(k)} n_2$, if*

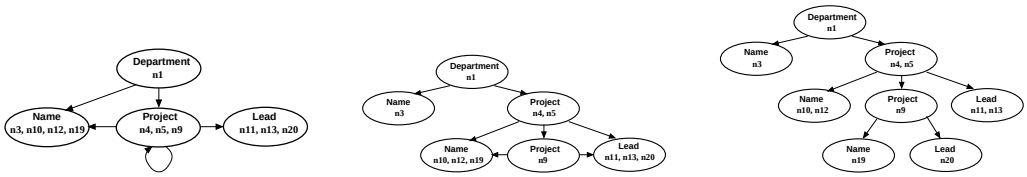
(1) $\lambda(n_1) = \lambda(n_2)$; and
(2) if $k \geq 1$ then
  (a) n_1 has a parent in D if and only if n_2 has a parent in D; and
  (b) if n_1 has parent p_1 and n_2 has parent p_2, then $p_1 \equiv_{A(k-1)} p_2$.

We call the partition induced by $\equiv_{A(k)}$ on V the $A(k)$-partition *of D.*

A more intuitive reading of this definition is that nodes n_1 and n_2 belong to the same block of the $A(k)$-partition if the label sequences associated with their incoming paths in D of length at most k are the same. Also note that the $A(k+1)$-partition of a document is a refinement of the $A(k)$-partition.

Example 6 *Figure 2 illustrates (ignoring for now the edges between the blocks), for $k = 0, 1,$ and 2, the $A(k)$-partition of the "Design" Department sub-tree rooted at node n_1 in the document of Figure 1.*

Following Kaushik et al. [13], the $A(k)$-*index* of a document D is a *graph* wherein each node is a block of the $A(k)$-partition of D, and an edge exists from a block B_1 to a block B_2 if there exists a parent-child edge in D from a node in B_1 to a node in B_2. So, the $A(k)$-index can be thought of as a representation of the $A(k)$-partition and how its blocks can be related in accordance with the document D. The $A(k)$-indexes for $k = 0, 1, 2$ are visualized in Figure 2 on the Design Department sub-tree of the document of Figure 1. Note that, if k is equal to the height of the document, then the $A(k)$-index corresponds to the 1-index proposed by Milo and Suciu [15] and the strong DataGuide proposed by Goldman and Widom [6].



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

Fig. 2. $A(k)$-indexes ($k = 0, 1, 2$) for the "Design" Department sub-tree rooted at node n_1 in the document of Figure 1.

2.3 The $P(k)$-partition of a document

The $A(k)$-partitions of a document D are partitions on its *nodes*. We will need another family of partitions, the $P(k)$-partitions, which, rather than being defined on nodes, are defined on the sets $\mathsf{UpPaths}(D, k)$, i.e., the sets of *upward-paths* of D of length at most k. As we will see, the $P(k)$-partitions are more fundamental than the $A(k)$-partitions for developing our results.

Definition 7 *Let D be a document, let $k \in \mathbb{N}$, and let (n_1, m_1) and (n_2, m_2) be two paths in $\mathsf{UpPaths}(D, k)$. We say that (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) $n_1 \equiv_{A(k)} n_2$; and
(2) $\mathsf{length}(n_1, m_1) = \mathsf{length}(n_2, m_2)$. [4]

We call the partition induced by $\equiv_{P(k)}$ on $\mathsf{UpPaths}(D, k)$ the $P(k)$-partition *of D.*

Example 8 *Consider the sub-tree D' in the document of Figure 1 rooted at n_4. For $k = 0, 1$, and 2, we have that*

(1) the $P(0)$-partition on D' is the set

$$\{[(n_{19}, n_{19}), (n_{10}, n_{10})],\ [(n_{20}, n_{20}), (n_{11}, n_{11})],\ [(n_9, n_9), (n_4, n_4)]\}.$$

(2) the $P(1)$-partition on D' is the set

$$\{[(n_{19}, n_{19}), (n_{10}, n_{10})],\ [(n_{20}, n_{20}), (n_{11}, n_{11})],\ [(n_9, n_9)], [(n_4, n_4)],\ [(n_{19}, n_9), (n_{10}, n_4)],\ [(n_{20}, n_9), (n_{11}, n_4)],\ [(n_9, n_4)]\}.$$

Notice that the block $[(n_9, n_9), (n_4, n_4)]$ of the $P(0)$-partition is split into two blocks of the $P(1)$-partition, namely $[(n_9, n_9)]$ and $[(n_4, n_4)]$. This is because $n_9 \equiv_{A_0} n_4$, but $n_9 \not\equiv_{A_1} n_4$.

[4] As should be clear, $\mathsf{length}(n, m)$ denotes the length of the path in D from node n to node m.

Table 1
The XPath-algebra path-semantics

$\emptyset(D)$	$= \emptyset$
$\varepsilon(D)$	$= \{(n,n) \mid n \in V\}$
$\hat{\ell}(D)$	$= \{(n,n) \mid n \in V \ \&\ \lambda(n) = \ell\}\ (\ell \in \mathcal{L})$
$\downarrow(D)$	$= Ed$
$\uparrow(D)$	$= Ed^{-1}$
$E_1 \cup E_2(D)$	$= E_1(D) \cup E_2(D)$
$E_1 \cap E_2(D)$	$= E_1(D) \cap E_2(D)$
$E_1 - E_2(D)$	$= E_1(D) - E_2(D)$
$E_1 \,;\, E_2(D)$	$= \{(n,m) \mid \exists p\colon (n,p) \in E_1(D)\ \&\ (p,m) \in E_2(D)\}$
$E_1[E_2](D)$	$= \{(n,m) \in E_1(D) \mid \exists p\colon (m,p) \in E_2(D)\}$

(3) the $P(2)$-partition on D' is the set

$$\{[(n_{19},n_{19})],\ [(n_{10},n_{10})],\ [(n_{20},n_{20})],\ [(n_{11},n_{11})],\ [(n_9,n_9)],\ [(n_4,n_4)],\\ [(n_{19},n_9)],\ [(n_{10},n_4)],\ [(n_{20},n_9)],\ [(n_{11},n_4)],\ [(n_9,n_4)],\\ [(n_{19},n_4)],\ [(n_{20},n_4)]\}.$$

Each $P(2)$-partition block is a singleton.

We observe that when k is equal to the height of a document D, then the $P(k)$-partition corresponds to the partitions induced by the 2-index on D proposed by Milo and Suciu [15].

2.4 The XPath-algebra

We present an algebraization [9] of the logical navigational core of XPath [7] which we adopt in this paper and define the paths and nodes-semantics of expressions in this algebra.

Definition 9 *The* XPath-algebra *consists of the primitives $\emptyset$, ε, $\hat{\ell}$ ($\ell \in \mathcal{L}$), $\downarrow$, and $\uparrow$, together with the operations on expressions $E_1 \,;\, E_2$, $E_1[E_2]$, $E_1 \cup E_2$, $E_1 \cap E_2$, and $E_1 - E_2$. Given a document $D = (V, Ed, r, \lambda)$, the* semantics *of an XPath-algebra expression E on D, denoted $E(D)$, is a subset of $\mathsf{Paths}(D)$. The semantics for each primitive and each operation is given in Table 1.*

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 at a particular node, as

follows.

Definition 10 *Let E be an XPath-algebra expression and let $D = (V, Ed, r, \lambda)$ be a document. For $n \in V$, the* local semantics *of E on D at n, denoted $E(D)(n)$, is the set $\{m \in V \mid (n, m)) \in E(D)\}$.*

Example 11 *Consider* /Projects/Department/Project[./Project]*, which is an XPath query that retrieves all the projects of departments that have a sub-project. When applied to the document D of Figure 1, this query returns the set of nodes $\{n_4, n_6\}$. An XPath-algebra expression corresponding to this query can be formulated as*[5] Projects;↓;Department;↓;Project[↓; Project]*. According to the semantics of the XPath-algebra, the global semantics of this expression on D is the set of paths $\{(n_0, n_4), (n_0, n_6)\}$ whereas its local semantics at the root node n_0 is the set of nodes $\{n_4, n_6\}$, which, as intended, corresponds to the result set of the original XPath query.*

2.5 Linking the $P(k)$-partition to the XPath-algebra

The $A(k)$-indexes were introduced to support efficient evaluation of certain path queries on XML documents. As was demonstrated empirically on a benchmark of queries, the performance benefits of these indexes were most dramatic when the queries "matched" the index definitions [13]. However, in that paper the concept of queries matching indexes was not formalized. A main theme of this paper is that we can indeed formalize this concept. More specifically, in the remainder of this section, we identify a class $\mathscr{U}(k)$ of sub-algebras of the XPath-algebra whose queries ideally match up with the $P(k)$-partitions (and as such with the $A(k)$ indexes). The central idea behind this formalization comes from showing that the $P(k)$-partitions are identical to the partitions induced on the document by the $\mathscr{U}(k)$-algebras. These language induced partitions are defined using equivalence relations that define a pair of paths equivalent when they can not be distinguished by the queries of the sub-algebras, i.e., they are either both in the answer of a query, or they are both not. (Compare with our motivating example in Section 1.1.) Intuitively, such pairs are always processed together during query evaluation.

In the balance of this section, we define the $\mathscr{U}(k)$-algebras and show how the $A(k)$ and $P(k)$-partitions are identical to partitions induced by these algebras.

[5] In practical examples, we shall systematically omit the "hat" of the labeling operator.

2.6 The $\mathscr{U}(k)$-algebras and their associated $\mathscr{U}(k)$-partitions

In the example of Section 1.1, we considered the class of relational queries RangeQ and introduced the notion of RangeQ-partitions. In this section, we define the $\mathscr{U}(k)$-algebras, and then, in analogy with this example, define the associated $\mathscr{U}(k)$-partitions.

Definition 12 *The* upward-k XPath algebras, *$\mathscr{U}(k)$, $k \in \mathbb{N}$, are recursively defined, as follows.*

(1) $\mathscr{U}(0)$ is the set of XPath-algebra expressions in which "$\downarrow$" and "$\uparrow$" primitives do not occur.
(2) For $k \geq 1$, $\mathscr{U}(k)$ is the smallest set of expressions satisfying
  (a) if $E \in \mathscr{U}(k-1)$, then $E \in \mathscr{U}(k)$;
  (b) $\uparrow \ \in \mathscr{U}(k)$;
  (c) if $E_1 \in \mathscr{U}(k_1)$ and $E_2 \in \mathscr{U}(k_2)$, and $k_1 + k_2 = k$, then $E_1 ; E_2 \in \mathscr{U}(k)$ and $E_1[E_2] \in \mathscr{U}(k)$; and
  (d) if $E_1 \in \mathscr{U}(k)$ and $E_2 \in \mathscr{U}(k)$, then $E_1 \cup E_2 \in \mathscr{U}(k)$, $E_1 \cap E_2 \in \mathscr{U}(k)$, and $E_1 - E_2 \in \mathscr{U}(k)$.

In particular, statement (2), (c), yields that, for $E \in \mathscr{U}(k-1)$, $E ; \uparrow$, $\uparrow ; E$, $E[\uparrow]$, and $\uparrow[E]$ are all in $\mathscr{U}(k)$.

Notice that the $\downarrow$ primitive cannot be used in $\mathscr{U}(k)$-algebra expressions.

Example 13 *The XPath-algebra expression* `Name` $; \uparrow ;$ `Project` $; \uparrow ;$ `Project` *is in $\mathscr{U}(2)$, but not in $\mathscr{U}(1)$. Similarly, $\uparrow ;$* `Department` *is in $\mathscr{U}(1)$, but not in $\mathscr{U}(0)$. Hence, if we combine both expressions as*

$$\texttt{Name} ; \uparrow ; \texttt{Project} ; \uparrow ; \texttt{Project}[\uparrow ; \texttt{Department}],$$

then this last expression is in $\mathscr{U}(3)$, but not in $\mathscr{U}(2)$.

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

Proposition 14 *Let D be a document, $k \in \mathbb{N}$, and $E \in \mathscr{U}(k)$. Then $E(D) \subseteq$ UpPaths(D, k).*

We are now ready to define the partitions associated with the $\mathscr{U}(k)$-algebras. Proposition 14 motivates us to define these partitions on UpPaths(D, k), just as we did for the $P(k)$-partitions.

Recall from the relational example in Section 1.1 that we associated the RangeQ query language with the RangeQ-partition. This partition was defined such that each of its blocks grouped those tuples in a relation that could not

be distinguished by the queries in RangeQ. We define the partitions associated with the $\mathscr{U}(k)$-algebras analogously, following Gyssens et al. [9].

Definition 15 *Let $D = (V, Ed, r, \lambda)$ be a document and $k \in \mathbb{N}$. We say two paths (n_1, m_1) and (n_2, m_2) in $\mathsf{UpPaths}(D, k)$ are $\mathscr{U}(k)$-*equivalent*, denoted $(n_1, m_1) \equiv_{\mathscr{U}(k)} (n_2, m_2)$, if, for any expression E in $\mathscr{U}(k)$, it is the case that $(n_1, m_1) \in E(D)$ if and only if $(n_2, m_2) \in E(D)$. We call the partition induced by $\equiv_{\mathscr{U}(k)}$ on $\mathsf{UpPaths}(D, k)$ the $\mathscr{U}(k)$-*partition of D.*

We next establish that the $P(k)$- and the $\mathscr{U}(k)$-partitions of D actually coincide.

2.7 The coupling of $P(k)$ and $\mathscr{U}(k)$

We proceed in two steps. First, we show in Lemma 17 that the $P(k)$-partition is at least as fine as the $\mathscr{U}(k)$-partition. By bootstrapping Lemma 17, we will then show in Theorem 18 that the two partitions coincide.

To show Lemma 17, we first need a technical lemma.

Lemma 16 *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 $\mathsf{length}(n_1, m_1) \leq k$. 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 $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$. Furthermore, $m_1 \equiv_{A(k-\mathsf{length}(n_1,m_1))} m_2$.*

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, respectively, 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), $\mathsf{length}(p_1, m_1) \leq k - 1$, and, by the induction hypothesis, there exists an ancestor m_2 of p_2 such that $(p_1, m_1) \equiv_{P(k-1)} (p_2, m_2)$ and $m_1 \equiv_{A(k-1-\mathsf{length}(p_1,m_1))} m_2$. It follows immediately that $(n_1, m_1) \equiv_{P(\mathsf{length}(n_1,m_1))} (n_2, m_2)$ and $m_1 \equiv_{A(k-\mathsf{length}(n_1,m_1))} m_2$.

We are now ready to show that the $P(k)$-partition is at least as fine as the $\mathscr{U}(k)$-partition.

Lemma 17 *Let $D = (V, Ed, r, \lambda)$ be a document, $k \in \mathbb{N}$, and let n_1, m_1, $n_2, m_2 \in V$ be such that m_1 is an ancestor of n_1, m_2 is an ancestor of n_2, and $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$. Then $(n_1, m_1) \equiv_{\mathscr{U}(k)} (n_2, m_2)$.*

PROOF. Let $E \in \mathscr{U}(k)$ and let D be a document. By symmetry, it suffices to prove that if $(n_1, m_1) \in E(D)$, then also $(n_2, m_2) \in E(D)$.

First, observe that it follows from $E \in \mathscr{U}(k)$ and $(n_1, m_1) \in E(D)$ that $\mathsf{length}(n_1, m_1) \leq k$, by Proposition 14.

The proof is a nested induction. The outer induction is on the value of k. The base case, $k = 0$, follows straightforwardly from the definition of $P(0)$-equivalence and a simple structural induction on expressions in $\mathscr{U}(0)$. Hence, we may assume that $k \geq 1$, and that the statement holds for $0, \ldots, k-1$.

The inner induction is a structural induction on expressions in $\mathscr{U}(k)$. By Definition 12, such expressions are built from expressions in $\mathscr{U}(k-1)$ and "$\uparrow$." We first consider these base cases of the structural induction.

- $E \in \mathscr{U}(k-1)$. The statement holds by the outer 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 $(n_2, m_2) \in \uparrow(D)$.

We have now established the basis of the inner, structural induction. Hence, we may additionally assume that $E \in \mathscr{U}(k)$ is neither an expression in $\mathscr{U}(k-1)$ nor "$\uparrow$." The following cases remain to be considered.

- $E = E_1 \cup E_2$, 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.
- $E = E_1; E_2$, for $E_1 \in \mathscr{U}(k_1)$ and $E_2 \in \mathscr{U}(k_2)$, such that $k_1 + k_2 = k$. Suppose $(n_1, m_1) \in E(D)$. Then there is a node $p_1 \in V$ such that $(n_1, p_1) \in E_1(D)$ and $(p_1, m_1) \in E_2(D)$. By Proposition 14, $\mathsf{length}(n_1, p_1) \leq k_1 \leq k$. By Lemma 16, there is a node $p_2 \in V$ such that $(n_1, p_1) \equiv_{P(k)} (n_2, p_2)$, and $p_1 \equiv_{A(k - \mathsf{length}(n_1, p_1))} p_2$. Since $k_1 \leq k$, it follows that also $(n_1, p_1) \equiv_{P(k_1)} (n_2, p_2)$. It follows either from the outer induction hypothesis (if $k_1 < k$) or from structural induction (if $k_1 = k$) that $(n_2, p_2) \in E_1(D)$. Since $k = k_1 + k_2$ and $\mathsf{length}(n_1, p_1) \leq k_1$, it follows that $k_2 \leq k - \mathsf{length}(n_1, p_1)$. Hence $p_1 \equiv_{A(k_2)} p_2$, whence $(p_1, m_1) \equiv_{P(k_2)} (p_2, m_2)$. It follows either from the outer induction hypothesis (if $k_2 < k$) or from structural induction (if $k_2 = k$) that $(p_2, m_2) \in E_2(D)$. Hence, $(n_2, m_2) \in E(D)$.
- $E = E_1[E_2]$, for $E_1 \in \mathscr{U}(k_1)$ and $E_2 \in \mathscr{U}(k_2)$, such that $k_1 + k_2 = k$. Suppose $(n_1, m_1) \in E(D)$. Then there is a node $p_1 \in V$ such that $(n_1, m_1) \in E_1(D)$ and $(m_1, p_1) \in E_2(D)$. By Proposition 14, $\mathsf{length}(n_1, m_1) \leq k_1$ and $\mathsf{length}(p_1, m_1) \leq k_2$. Since $k_1 \leq k$, it follows from $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$ that $(n_1, m_1) \equiv_{P(k_1)} (n_2, m_2)$. From $(n_1, m_1) \in E_1(D)$, it now follows either

from the outer induction hypothesis (if $k_1 < k$) or from structural induction (if $k_1 = k$) that $(n_2, m_2) \in E_1(D)$. From a straightforward inductive argument, it follows from $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$ that $m_1 \equiv_{A(k-\mathsf{length}(n_1,m_1))} m_2$. As in the previous case, we derive that $m_1 \equiv_{A(k_2)} m_2$. By Lemma 16, there is a node $p_2 \in V$ such that $(m_1, p_1) \equiv_{P(k_2)} (m_2, p_2)$. It follows either from the outer induction hypothesis (if $k_2 < k$) or from structural induction (if $k_2 = k$) that $(m_2, p_2) \in E_2(D)$. Hence, $(n_2, m_2) \in E(D)$.

We now establish the following coupling theorem for the $P(k)$ and $\mathscr{U}(k)$ partitions, in analogy to Proposition 1.

Theorem 18 [Coupling Theorem]

Let $D = (V, Ed, r, \lambda)$ be a document and $k \in \mathbb{N}$. The $P(k)$-partition of D and the $\mathscr{U}(k)$-partition of D coincide.

PROOF. Let $n_1, m_1,\ n_2, m_2 \in V$ such that m_1 is an ancestor of n_1 and m_2 is an ancestor of n_2. Let B be the block of the $P(k)$-partition containing both (n_1, m_1) and (n_2, m_2). By Lemma 17, $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$ implies $(n_1, m_1) \equiv_{\mathscr{U}(k)} (n_2, m_2)$. It remains to show that $(n_1, m_1) \not\equiv_{P(k)} (n_2, m_2)$ implies $(n_1, m_1) \not\equiv_{\mathscr{U}(k)} (n_2, m_2)$. Thereto, it suffices to exhibit a labeling query $\mathtt{label}_B \in \mathscr{U}(k)$ that satisfies $\mathtt{label}_B(D) = B$, since $(n_1, m_1) \in \mathtt{label}_B(D)$ and $(n_2, m_2) \notin \mathtt{label}_B(D)$ does indeed imply that $(n_1, m_1) \not\equiv_{\mathscr{U}(k)} (n_2, m_2)$. The expression $\mathtt{label}_B$, the construction of which is rather involved, is also of independent interest. Therefore, we defer the construction of this labeling expression to Section 4, which is entirely devoted to this issue.

As an immediate corollary, each $\mathscr{U}(k)$ query evaluated on a document D is equal to the union of a family of blocks of the $P(k)$-partition of D.

Corollary 19 [Block-Union Theorem]

Let D be a document, $k \in \mathbb{N}$, and $E \in \mathscr{U}(k)$. Then there exists a class $\mathcal{B}_E$ of partition blocks of the $P(k)$-partition of D such that

$$E(D) = \bigcup_{B \in \mathcal{B}_E} B.$$

In the remainder of this paper, we will focus on the consequences of the Coupling Theorem for query processing. In particular, we will see that in analogy with Proposition 2, the Block-Union Theorem provides insight into the processing of general XPath-algebra queries. We will return to discuss the relative merits of index structures built on the $A(k)$ and $P(k)$ partitions in Section 5.

2.8 The coupling of $A(k)$ and $\mathscr{U}(k)$

To conclude Section 2, we also explore the relationship between the $A(k)$-partitions and a notion of indistinguishability under the "local" semantics of $\mathscr{U}(k)$ expressions evaluated at particular nodes in a document (Definition 10), following the methodology of Gyssens et al. [9]. We contrast the results obtained below with the results of Section 2.7, and argue that, from the viewpoint of query processing, it is really the $P(k)$- and not the $A(k)$-partitions that matter.

We first consider a notion of $\mathscr{U}(k)$-indistinguishability at the node level that can be compared with $A(k)$-equivalence.

Definition 20 *Let $D = (V, Ed, r, \lambda)$ be a document and $k \in \mathbb{N}$. Two nodes $n_1, n_2 \in V$ are* $\mathscr{U}(k)$-equivalent*, denoted $n_1 \equiv_{\mathscr{U}(k)} n_2$, if, for every expression E in $\mathscr{U}(k)$, it is the case that $E(D)(n_1) = \emptyset$ if and only if $E(D)(n_2) = \emptyset$.*

We now show the following:

Theorem 21 *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 $n_1 \equiv_{A(k)} n_2$.*

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 $(n_1, m_1) \in E(D)$. By Lemma 16, there exists $m_2 \in V$ such that $(n_1, m_1) \equiv_{P(k)} (n_2, m_2)$. By Lemma 17, $(n_2, m_2) \in E(D)$, 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 $\widehat{\lambda(n_1)} \in \mathscr{U}(k)$. Then $\widehat{\lambda(n_1)}(D)(n_1) \neq \emptyset$ and $\widehat{\lambda(n_1)}(D)(n_2) = \emptyset$, 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 finish the proof of the statement of the theorem by induction on k. For the base case $k = 0$, this follows immediately from the first fact above.

Assume therefore that $k \geq 1$, and that the statement holds for $k - 1$. If n_1

and n_2 are both the root, then, trivially, the statement holds for k as well. Otherwise, by the second fact above, n_1 and n_2 have both parents, say p_1 and p_2, respectively. Suppose p_1 and p_2 are not $\mathscr{U}(k-1)$-equivalent. Then, there exists an expression E in $\mathscr{U}(k-1)$ such that, e.g., $E(D)(p_1) \neq \emptyset$ and $E(D)(p_2) = \emptyset$. Now consider the expression $F = \uparrow ; E$, which is in $\mathscr{U}(k)$. Clearly, $F(D)(n_1) \neq \emptyset$ and $F(D)(n_2) = \emptyset$, 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 this result and the first fact above, the theorem immediately follows.

One might be tempted to compare Theorem 21 with the Coupling Theorem (Theorem 18) in Section 2.7, and, therefore, wonder why we do not present here an analog of the Block-Union Theorem (Corollary 19). Unfortunately, this comparison does not hold. In Section 2.7, the $\mathscr{U}(k)$-partitions categorize possible *outputs* of $\mathscr{U}(k)$ expressions under the "global" semantics. Here, the $\mathscr{U}(k)$-partitions do *not* categorize possible outputs of $\mathscr{U}(k)$ expressions under the "local" semantics. Rather, they categorize nodes which are part of the *input*. Because of this, Theorem 21 cannot be characterized as a coupling theorem, and as a consequence it can not be bootstrapped to a block-union theorem.

3 XPath query evaluation with $P(k)$-partitions

The results of Section 2 deal with answering $\mathscr{U}(k)$ queries directly using index structures based on the $P(k)$-partition. In this section, we consider the evaluation of more general XPath algebra expressions and show how the results of Section 2 concerning the coupling between the $\mathscr{U}(k)$ and $P(k)$-partitions can be utilized in this case. Given an XPath expression and a $P(k)$-partition, the main idea is to identify its $\mathscr{U}(k)$ sub-expressions or those that are easily converted to $\mathscr{U}(k)$ expressions using rewrite rules. For each such expression, we are then guaranteed by the Block-Union Theorem that its value is the union of an appropriate set of blocks of the $P(k)$-partition. If we then have a method to quickly identify and return partition blocks, we will have an efficient way of evaluating these expressions. We return to this last issue in the next section. In this section, we focus on the development of general techniques for using $P(k)$-partitions in the evaluation of XPath algebra expressions.

3.1 Evaluating Upward Expressions

If our given XPath expression is in fact a member of $\mathscr{U}(k)$ then no decomposition is necessary. However, if we consider a $\mathscr{U}(j)$ expression where $j > k$, then

such a query is not directly supported by the $P(k)$-partition. Nevertheless, we can decompose it into sub-expressions that are in $\mathscr{U}(k)$. For example, suppose that the $P(2)$-partition of the example document D in Figure 1 is available and we have the expression

$$E = \texttt{Lead} \,;\uparrow;\, \texttt{Project} \,;\uparrow;\, \texttt{Project} \,;\uparrow[\texttt{Department} \,;\uparrow;\, \texttt{Projects}]$$

in $\mathscr{U}(4)$. Then E contains the sub-expressions

$$G_1 = \texttt{Lead} \,;\uparrow;\, \texttt{Project} \,;\uparrow;\, \texttt{Project}, \text{ and}$$
$$G_2 = \texttt{Project} \,;\uparrow[\texttt{Department} \,;\uparrow;\, \texttt{Projects}]$$

which are both in $\mathscr{U}(2)$. As such, they can be directly evaluated using the $P(2)$-partition as $E_1(D) = G_1(D) \bowtie G_2(D)$.

3.2 Evaluating Downward Expressions

In practice, most XPath expressions use navigation just along the parent-child ($\downarrow$) axis. Consider the XPath sub-algebra $\mathscr{D}$ which is defined as the set of all XPath expressions in which the $\uparrow$ primitive does not appear. For such queries, we cannot directly utilize the Block-Union Theorem. However, we can convert downward navigation into upward navigation by using a technique which we will refer to as "inverting expressions." We illustrate this technique on downward expressions with and without predicate operations. For this discussion, we consider downward expressions to be in the $\mathscr{D}(k)$-algebra which is defined in complete analogy with $\mathscr{U}(k)$, except that the $\downarrow$ primitive is permitted, but not the $\uparrow$ primitive.

3.2.1 Downward expressions without predicates

Downward expressions without predicates can be "inverted" into corresponding upward expressions without predicates using the rewrite rules shown in Table 2.

So, given a downward expression $E \in \mathscr{D}(k)$ without predicates, we can rewrite E into E^{-1} which is in $\mathscr{U}(k)$ and also has no predicates. Given a document D, we can then obtain $E(D)$ by first computing $E^{-1}(D)$ and then inverting the result. Since E^{-1} is an expression in $\mathscr{U}(k)$, we can directly apply the evaluation techniques for $\mathscr{U}(k)$ expressions discussed above.

Table 2
Inversion Rewrite Rules for downward expressions without predicates.

E	E^{-1}
$\emptyset$	$\emptyset$
ε	ε
$\hat{\ell}$	$\hat{\ell}$
$\downarrow$	$\uparrow$
$E_1 \cup E_2$	$E_1^{-1} \cup E_2^{-1}$
$E_1 \cap E_2$	$E_1^{-1} \cap E_2^{-1}$
$E_1 - E_2$	$E_1^{-1} - E_2^{-1}$
$E_1 \,;\, E_2$	$E_2^{-1} \,;\, E_1^{-1}$

3.2.2 Downward expressions with predicates

Now consider the evaluation of downward algebra expressions wherein predicate operations occur. A simple example is the expression $E = \downarrow[\downarrow]$. Applied to a document, E evaluates to the document's parent-child pairs for children that have at least one child themselves. As above, to evaluate E on a document D, we could consider the concept of inverting E into an expression $E^{-1} \in \mathscr{U}(2)$ such that $E(D) = (E^{-1}(D))^{-1}$. This approach does not work here because the inversion rules in Table 2 unfortunately do not extend to include the predicate operation, as can be derived from the following result.

Proposition 22 *Let $D = (V, Ed, r, \lambda)$ be a document with $V = \{n_a, n_b, n_c, n_d\}$, $Ed = \{(n_a, n_b), (n_b, n_c), (n_c, n_d)\}$, $r = n_a$, and $\lambda(n_a) = \lambda(n_b) = \lambda(n_c) = \lambda(n_d)$, as shown in Figure 3. Then, for each expression $G \in \mathscr{U}(2)$, we have that (n_c, n_b) and (n_d, n_c) are both in $G(D)$ or both not in $G(D)$.*

PROOF. The $A(0)$-partition on V is $\{[n_a, n_b, n_c, n_d]\}$, since all nodes have the same label. The $A(1)$-partition on V is $\{[n_a], [n_b, n_c, n_d]\}$, since n_a has no parent, and the $A(2)$-partition on V is $\{[n_a], [n_b], [n_c, n_d]\}$, since n_b has no grandparent. It follows that the subset of the $P(2)$-partition involving pairs of nodes of length exactly 1 is $\{[(n_b, n_a)], [(n_c, n_b), (n_d, n_c)]\}$. By the Block Union Theorem (Corollary 19), $G(D)$ is a union of blocks of the $P(2)$-partition of D. Since (n_c, n_b) and (n_d, n_c) are in the same block they are either both in or both not in $G(D)$.

Notice that a direct proof of Proposition 22, though feasible, requires a lengthy and tedious case analysis.

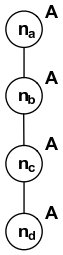


Fig. 3. The document D in the statement of Proposition 22.

We now return to our expression $E = \downarrow[\downarrow]$. For the document D of Proposition 22, $E^{-1}(D) = \{(n_b, n_a), (n_c, n_b)\}$. By Proposition 22, we have that for *each* expression $G \in \mathscr{U}(2)$, $G(D) \neq E(D)^{-1}$, whence E^{-1} is *not* equivalent to an expression in $\mathscr{U}(2)$.

We can however use another technique to deal with such expressions, which is based on the following result.

Proposition 23 *Let $k \in \mathbb{N}$. Each boolean-free expression E in the $\mathscr{D}(k)$-algebra (i.e, without union, intersection, or difference) can be normalized to an expression of the form*

$$F_1 \,;\, G_1 \,;\, F_2 \,;\, G_2 \,;\, \ldots \,;\, F_{n-1} \,;\, G_{n-1} \,;\, F_n \,;\, G_n \,,$$

where

(1) $F_1, F_2, \ldots, F_{n-1}, F_n$ are boolean-free $\mathscr{D}(k)$ expressions in which no predicates occur;
(2) $F_2, \ldots, F_{n-1}, F_n$ are not $\mathscr{D}(0)$ expressions;
(3) $G_1, G_2, \ldots, G_{n-1}, G_n$ are boolean-free $\mathscr{U}(k)$ expressions in which no predicates occur; and
(4) $G_1, \ldots, G_{n-1}$ are not $\mathscr{U}(0)$ expressions.

PROOF.

The proof is a nested induction. The outer induction is on the value of k. For the base case, $k = 0$, it suffices to observe that each boolean-free $\mathscr{D}(0)$ expression is equivalent to one of the primitives $\emptyset$, ε, or $\hat{\ell}$ ($\ell \in \mathcal{L}$), as can be shown by a simple structural induction. Hence, we may assume that $k \geq 1$, and that the proposition holds for $0, \ldots, k-1$.

The inner induction is a structural induction on expressions in $\mathscr{D}(k)$, which are built from expressions in $\mathscr{D}(k-1)$ and "$\downarrow$." Clearly, the proposition holds for these base cases of the structural induction. Hence, we may additionally assume that $E \in \mathscr{D}(k)$ is neither an expression in $\mathscr{D}(k-1)$ nor "$\downarrow$." The following cases remain to be considered.

- $E = E_1; E_2$ with $E_1 \in \mathscr{D}(k_1)$, $E_2 \in \mathscr{D}(k_2)$, and $k_1 + k_2 = k$. By the outer induction hypothesis or the inner structural induction hypothesis (if k_1 or k_2 equals k), we may assume that

$$E_1 = F_{11}\,;G_{11}\,;F_{12}\,;G_{12}\,;\ldots;F_{1(m-1)}\,;G_{1(m-1)}\,;F_{1m}\,;G_{1m} \text{ and}$$
$$E_2 = F_{21}\,;G_{21}\,;F_{22}\,;G_{22}\,;\ldots;F_{2(n-1)}\,;G_{2(n-1)}\,;F_{2n}\,;G_{2n}$$

  satisfying the conditions of the proposition for k_1 and k_2, respectively. Hence,

$$E = F_{11}\,;G_{11}\,;F_{12}\,;G_{12}\,;\ldots;F_{1(m-1)}\,;G_{1(m-1)}\,;F_{1m}\,;G_{1m}$$
$$F_{21}\,;G_{21}\,;F_{22}\,;G_{22}\,;\ldots;F_{2(n-1)}\,;G_{2(n-1)}\,;F_{2n}\,;G_{2n}\,.$$

  If neither G_{1m} is in $\mathscr{U}(0)$ nor F_{21} is in $\mathscr{D}(0)$, then the above normalization satisfies the proposition. If G_{1m} is not in $\mathscr{U}(0)$, but F_{21} is in $\mathscr{D}(0)$, then $G_{1m}\,;F_{21}\,;G_{21}$ is in $\mathscr{U}(k)$. If $n \neq 1$, this expression is not in $\mathscr{U}(0)$. Finally, if G_{1m} is in $\mathscr{U}(0)$, then $F_{1m}\,;G_{1m}\,;F_{21}$ is in $\mathscr{D}(k)$. If $m \neq 1$, this expression is not in $\mathscr{U}(0)$.
- $E = E_1[E_2]$ with $E_1 \in \mathscr{D}(k_1)$, $E_2 \in \mathscr{D}(k_2)$, and $k_1 + k_2 = k$. By the outer induction hypothesis or the inner structural induction hypothesis (if k_1 or k_2 equals k), we may assume that

$$E_1 = F_{11}\,;G_{11}\,;F_{12}\,;G_{12}\,;\ldots;F_{1(m-1)}\,;G_{1(m-1)}\,;F_{1m}\,;G_{1m} \text{ and}$$
$$E_2 = F_{21}\,;G_{21}\,;F_{22}\,;G_{22}\,;\ldots;F_{2(n-1)}\,;G_{2(n-1)}\,;F_{2n}\,;G_{2n}\,,$$

  satisfying the conditions of the proposition for k_1 and k_2, respectively. Now, it is easily seen that $E_1[E_2]$ is equivalent to $E_1\,;E_2\,;E_2^{-1}$.[6] Notice that E_2^{-1} can be normalized as

$$E_2^{-1} = G_{2n}^{-1}\,;F_{2n}^{-1}\,;G_{2(n-1)}^{-1}\,;F_{2(n-1)}^{-1}\,;\ldots;G_{22}^{-1}\,;F_{22}^{-1}\,;G_{21}^{-1}\,;F_{21}^{-1}\,.$$

  Since all the subexpressions above are predicate-free, the inverted $\mathscr{D}(k)$ expressions $F_{2n}^{-1}, F_{2(n-1)}^{-1}, \ldots, F_{22}^{-1}, F_{21}^{-1}$ can be converted to $\mathscr{U}(k)$ expressions according to the rules in Table 2. Similarly, the inverted $\mathscr{U}(k)$ expressions $G_{2n}^{-1}, G_{2(n-1)}^{-1} \ldots, G_{22}^{-1}, G_{21}^{-1}$ can be converted to $\mathscr{D}(k)$ expressions. Hence, the normalization for E_2^{-1} satisfies the proposition. The remainder of this case now follows immediately from the case for composition.

So, every boolean-free $\mathscr{D}(k)$ expression can be written as an alternating composition of boolean-free $\mathscr{D}(k)$ and $\mathscr{U}(k)$ expressions in which no predicates

[6] For general XPath-algebra expressions E_1 and E_2, $E_1[E_2]$ is equivalent to $E_1; (E_2; E_2^{-1} \cap \varepsilon)$. Since E_2 is a downward XPath-algebra expression, we can simplify $E_2; E_2^{-1} \cap \varepsilon$ to $E_2; E_2^{-1}$.

occur. If we apply Proposition 23 to the expression $E = \downarrow[\downarrow]$, we find that it is equivalent to the XPath algebra expression $\downarrow ; \downarrow ; \uparrow$. Its sub-expression $F = \downarrow ; \downarrow$ is in $\mathscr{D}(2)$, and its sub-expression $G = \uparrow$ is in $\mathscr{U}(1)$. By applying the inversion technique described in Section 3.2.1 to F, the evaluation of $E(D)$ can be accomplished by computing the relation $(F^{-1}(D))^{-1} \bowtie G(D)$, and, as indicated earlier in this section, the evaluations of $G(D) = \uparrow(D)$ and $F^{-1}(D) = \uparrow ; \uparrow(D)$ can be done by utilizing the Block-Union Theorem for the $P(2)$- and $P(1)$-partitions, respectively.

Given that the selectivity of a longer path is no larger than that of short subpaths of the path, evaluating F^{-1} reduces the search space to the minimum that can be obtained on such a chain expression. It is reasonable to claim that, generally speaking, the result of $F^{-1}(D)$ is substantially smaller than that of $G(D)$, and, hence, the $\bowtie$ operation can be further optimized as $F^{-1}(D)$ followed by an upward navigation.

Finally, we must observe that, in the worst case, the size of the normalized expression in Proposition 23 can be exponential in the size of the original expression. However, as we shall argue in Section 5, the $P(k)$-indexes considered in practice will typically have small values for k, so the above observation is not a real drawback.

We also want to observe that boolean operations are best dealt with by standard relational query processing techniques.

In the last example of this section, we will now consider a slightly more complicated expression.

Example 24 *E in $\mathscr{D}(3)$, applied to the document D of Figure 1. This expression retrieves information about leaders of projects that have a sub-project:*

$$E = \texttt{Department} ; \downarrow ; \texttt{Project}[\downarrow ; \texttt{Project}] ; \downarrow ; \texttt{Lead}.$$

If we apply Proposition 23 to this expression, we find

$$E = F_2 ; F_5^{-1} ; F_4,$$

with[7]

$$F_2 = \texttt{Department} ; \downarrow ; \texttt{Project} ; \downarrow ; \texttt{Project},$$

$$F_5 = \texttt{Project} ; \downarrow ; \texttt{Project}, \text{ and}$$

$$F_4 = \texttt{Project} ; \downarrow ; \texttt{Lead}.$$

[7] The strange numbering of the expressions is to be consistent with Figure 4 and the example to which this figure pertains.

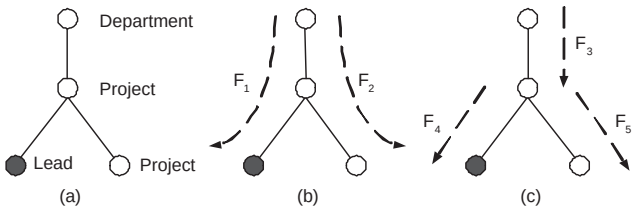


Fig. 4. Chain pattern tree for the expression E in Example 24.

Another way of going about the expression E is representing it as an expression pattern tree, as illustrated in Figure 4, (a). The shaded node can be interpreted as the "answer" of E applied to the root of the pattern.

Assume that the $P(2)$-partition is available on D. Then, as shown in Figure 4, (b), there are two natural chains of length 2 present in the pattern tree of E: F_1 and F_2. There are also natural chains of length 1 as shown in Figure 4(c): F_3, F_4, and F_5. Each of these chains corresponds to a downward algebra expression:

$$F_1 = \texttt{Department} \,;\downarrow;\, \texttt{Project} \,;\downarrow;\, \texttt{Lead}\,,$$

$$F_2 = \texttt{Department} \,;\downarrow;\, \texttt{Project} \,;\downarrow;\, \texttt{Project}\,,$$

$$F_3 = \texttt{Department} \,;\downarrow;\, \texttt{Project}\,,$$

$$F_4 = \texttt{Project} \,;\downarrow;\, \texttt{Lead}\,, \text{ and}$$

$$F_5 = \texttt{Project} \,;\downarrow;\, \texttt{Project}\,.$$

(Notice that we already encountered F_2, F_4, and F_5.) Using F_1, F_2, and F_4, we can rewrite the expression E as

$$E = ((F_1 \,;\uparrow) \cap (F_2 \,;\uparrow)) \,; F_4,$$

and, therefore, $E(D)$ can be computed as[8]

$$\{[(F_1^{-1}(D))^{-1} \bowtie \uparrow(D)] \cap [(F_2^{-1}(D))^{-1} \bowtie \uparrow(D)]\} \bowtie (F_4^{-1}(D))^{-1}.$$

All sub-expressions in this transformed expression of E are in $\mathscr{U}(2)$, and hence can be evaluated using the Block-Union Theorem for $P(2)$.

Now assume that only the $P(1)$-partition is available. In this case, the longest path expressions that can take advantage of the partitions are those of length at most 1. Such expressions are F_3, F_4, and F_5. Using these sub-expressions, E can be rewritten as[9]

$$E = \{[(F_3 \,; F_4) \,;\uparrow] \cap [(F_3 \,; F_5) \,;\uparrow]\} \,; F_4.$$

[8] Square brackets and curly brackets have been used to improve readability.
[9] Again, square brackets and curly brackets have been used to improve readability.

We have just observed how the Block-Union Theorem assists in the evaluation of XPath expressions. However, if we want to make efficient utilization of these ideas, we will need techniques for quickly identifying the $P(k)$-partition blocks associated with a query. We turn to this issue in the following section.

4 Labeling $P(k)$-partition blocks

In Section 2, we showed that the $\mathscr{U}(k)$-partition and the $P(k)$-partition coincide (Theorem 18). To complete the proof of this result, we still need to show the existence of *labeling queries* associated with the $P(k)$-partition. More concretely, we require that, for each partition block B of the $P(k)$-partition, there exists a $\mathscr{U}(k)$expression $\texttt{label}_B$ satisfying $\texttt{label}_B(D) = B$. This requirement yields a syntactic characterization of the $P(k)$-partition in terms of $\mathscr{U}(k)$ expressions, which complements the semantic relationship expressed in Corollary 19. This syntactic relationship is critical in identifying the $P(k)$-partition blocks used in query evaluation. In particular, we have that evaluation of a $\mathscr{U}(k)$ query on a document D can be done by forming a union of partition block labeling expressions applied to D, similarly to Proposition 3 for the range queries.

Theorem 25 [**Label-Union Theorem**]

Let D be a document and $k \in \mathbb{N}$. Then for each query $E \in \mathscr{U}(k)$, a set of labeling queries $\mathcal{L}_E \subseteq \mathscr{U}(k)$ can be constructed such that

$$E(D) = \bigcup_{\texttt{label} \in \mathcal{L}_E} \texttt{label}(D).$$

The Label-Union Theorem is an immediate corollary to Theorem 19, provided we can show the following:

> Let D be a document and $k \in \mathbb{N}$. For each block B of the $P(k)$-partition of D, an expression $\texttt{label}_B \in \mathscr{U}(k)$ can be constructed such that $\texttt{label}_B(D) = B$.

The remainder of this section is organized as follows. First, we define in two steps the labeling expressions for partition blocks. Then, in the third and final subsection, we make precise the relationship of these expressions to the partition blocks.

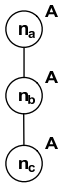


Fig. 5. The document D of Example 28.

4.1 Step 1: Ancestor path expressions

Given $k \in \mathbb{N}$, we first define *k-ancestor label expressions.*

Definition 26 *Let $k \in \mathbb{N}$, let $D = (V, Ed, r, \lambda)$ be a document, and let $n \in V$. Let the* k-ancestor label path *of n be the list of labels $\ell_0, \ldots, \ell_{k_n}$ of the nodes on the path from n up towards the root node r, of length $k_n = \min\{k, \mathsf{length}(n, r)\}$. We denote the k-ancestor label path of n as $\mathsf{alp}(k, n)$. For $i \leq k_n$, the i^{th}* k-ancestor label expression of n *is the $\mathscr{U}(k)$ expression*

$$L_{k,n,i} = \begin{cases} \hat{\ell}_0 \,;\uparrow;\, \hat{\ell}_1 \,;\ldots;\uparrow;\, \hat{\ell}_i[\uparrow;\, \hat{\ell}_{i+1} \,;\ldots;\uparrow;\, \hat{\ell}_{k_n}] & \text{if } i < k_n; \\ \hat{\ell}_0 \,;\uparrow;\, \hat{\ell}_1 \,;\ldots;\uparrow;\, \hat{\ell}_{k_n} & \text{if } i = k_n \end{cases}$$

The following observation follows directly from the definition of $P(k)$-equivalence and Definition 26.

Proposition 27 *Let $k \in \mathbb{N}$, let $D = (V, Ed, r, \lambda)$ be a document, and let B be a block of the $P(k)$-partition on D. Let $(n_1, m_1), (n_2, m_2) \in B$.*[10] *Then*

(1) $L_{k,n_1,\mathsf{length}(n_1,m_1)} = L_{k,n_2,\mathsf{length}(n_2,m_2)}$*; and*
(2) $B \subseteq L_{k,n_1,\mathsf{length}(n_1,m_1)}(D) = L_{k,n_2,\mathsf{length}(n_2,m_2)}(D)$.

So, all members of a $P(k)$ partition block share a k-ancestor label expression.

Example 28 *Consider the $P(1)$-partition of the small document in Figure 5, wherein each node has label* `A`*:*

$$\{[(n_a, n_a)],\ [(n_b, n_b), (n_c, n_c)],\ [(n_c, n_b), (n_b, n_a)]\}.$$

As noted in Proposition 27, we can associate with each block in this partition the expression $L_{1,n,\mathsf{length}(n,m)}$, where (n, m) is an arbitrary element of the block, as shown in Table 3.

Note, however, that ancestor label expressions do *not* necessarily *characterize* particular $P(k)$ blocks, in the sense that the inclusion in the second statement of Proposition 27 is *not* necessarily an equality.

[10] Notice that this implies that $\mathsf{length}(n_1, m_1) = \mathsf{length}(n_2, m_2)$.

Table 3
The shared 1-ancestor label expressions of the blocks of the $P(1)$-partition on the document of Figure 5.

Partition Block	Expression
$[(n_a, n_a)]$	$L_{1,n_a,0} = \mathtt{A}$
$[(n_b, n_b), (n_c, n_c)]$	$L_{1,n_b,0} = \mathtt{A}[\uparrow ; \mathtt{A}]$
$[(n_c, n_b), (n_b, n_a)]$	$L_{1,n_b,1} = \mathtt{A} ; \uparrow ; \mathtt{A}$

Example 29 *Continuing Example 28, we note that expression "*$\mathtt{A}$*" for block* $[(n_a, n_a)]$ *evaluates on the document* D *in Figure 5 as*

$$\mathtt{A}(D) = \{(n_a, n_a), (n_b, n_b), (n_c, n_c)\}$$

and hence does not characterize its block. In other words, $L_{1,n_a,0}$ *is not selective enough. In particular, all blocks, with* 1*-ancestor label expressions having* $L_{1,n_a,0}$ *as a* prefix *expression will also appear in the evaluation of* $L_{1,n_a,0}$*. For example, the 1-ancestor labeling expression* $\mathtt{A}[\uparrow ; \mathtt{A}]$ *for block* $[(n_b, n_b), (n_c, n_c)]$ *has as a prefix the 1-ancestor labeling expression* $\mathtt{A}$ *for block* $[(n_a, n_a)]$*, and, therefore, both blocks appear in the evaluation of* $\mathtt{A}$*.*

We pursue a remedy for this problem in the next step.

4.2 Step 2: Partition labeling expressions

To tighten up ancestor label expressions, we need two tools. To compare these expressions, we introduce the following notion of expression prefixes.

Definition 30 *Let* $k \in \mathbb{N}$*, let* $D = (V, Ed, r, \lambda)$ *be a document, and let* $n, n' \in V$*. We denote by* $\mathsf{alp}(k, n) \prec \mathsf{alp}(k, n')$ *that the* k*-ancestor label path of node* n *is a* strict prefix *of the* k*-ancestor label path of node* n'*.*

Example 31 *In Example 29, we observed that* $\mathsf{alp}(1, n_a) \prec \mathsf{alp}(1, n_b)$*.*

To precisely characterize a block of a $P(k)$-partition, we shall eliminate the spurious pairs introduced by the corresponding k-ancestor labeling expression $L_{k,n,i}$ by using all k-ancestor labeling expressions of which $L_{k,n,i}$ is a prefix.

Definition 32 *Let* $k \in \mathbb{N}$*, let* $D = (V, Ed, r, \lambda)$ *be a document, and let* $(n, m) \in \mathsf{UpPaths}(D, k)$*. Then the* k-partition labeling expression *for* (n, m)

is the $\mathscr{U}(k)$ expression

$$\texttt{label}_{k,(n,m)} \;=\; L_{k,n,l} \;-\; \bigcup_{\substack{n' \in V \\ \mathsf{alp}(k,n) \prec \mathsf{alp}(k,n')}} L_{k,n',l}\,,$$

where $l = \mathsf{length}(n,m)$.

Example 33 *We observed in Example 31 that $\mathsf{alp}(1,n_a) \prec \mathsf{alp}(1,n_b)$, whence $\texttt{label}_{1,(n_a,n_a)} = L_{1,n_a,0} - L_{1,n_b,0} = \texttt{A} - \texttt{A}[\uparrow;\texttt{A}]$. When we evaluate this expression on document D of Figure 5, this gives us precisely the $P(1)$-partition block to which the pair (n_a,n_a) belongs, i.e., $\texttt{label}_{1,(n_a,n_a)}(D) = [(n_a,n_a)]$, as desired.*

Below, we show that the technique described in Step 2 is adequate in general as well.

4.3 Relationship to Partition Blocks

We are now ready to show the following.

Proposition 34 *Let $k \in \mathbb{N}$, let $D = (V, Ed, r, \lambda)$ be a document, and let $(n,m) \in \mathsf{UpPaths}(D,k)$. Let B be the block of the $P(k)$-partition to which (n,m) belongs. Then $\texttt{label}_{k,(n,m)}(D) = B$.*

PROOF. For easy reference, let $l = \mathsf{length}(n,m)$ and let ℓ be the label of n.

We first show that $B \subseteq \texttt{label}_{k,(n,m)}(D)$. By Proposition 27, we know that $B \subseteq L_{k,n,l}$. Now, let n' be a node such that $\mathsf{alp}(k,n) \prec \mathsf{alp}(k,n')$. This is only possible if $k_n = \mathsf{length}(\mathsf{alp}(k,n)) = \mathsf{length}(n,r)$ and $k_{n'} = \mathsf{length}(\mathsf{alp}(k,n')) > \mathsf{length}(n,r)$. Hence, $(n,m) \notin L_{k,n',l}(D)$. By Proposition 27, this implies in turn that no member of B is in $L_{k,n',l}(D)$. By Definition 32, $B \subseteq \texttt{label}_{k,(n,m)}(D)$.

We now show that this inclusion is actually an equality. Thereto, we have to show, for $(n',m') \in \mathsf{UpPaths}(D,k)$ with $(n',m') \notin B$, that $(n',m') \notin \texttt{label}_{k,(n,m)}(D)$. First, we single out a few cases for which our assertion follows very easily.

- If $\mathsf{length}(n',m') \neq l$, then $(n',m') \notin L_{k,n,l}(D)$, for, by Definition 26, $L_{k,n,l}$ returns pairs of equal length. Consequently, $(n',m') \notin \texttt{label}_{k,(n,m)}(D)$. We may therefore assume in the remainder of this proof that $\mathsf{length}(n',m') = l$.
- If ℓ is not also the label of n', then, by Definitions 7 and 5, $(n,m) \not\equiv_{P(k)} (n',m')$. Also, by Definition 26, $(n',m') \notin L_{k,n,l}$. Consequently, $(n',m') \notin$

$\texttt{label}_{k,(n,m)}(D)$. We may therefore assume in the remainder of this proof that the label of n' is ℓ.

We now proceed by induction on k.

For the base case, $k = 0$, we have that, necessarily $n = m$ and $n' = m'$. By Definitions 7 and 5, (n', n') can belong to another block of the $P(0)$-partition if ℓ is not the label of n'. But this case has already been dealt with above.

We may thus assume that $k \geq 1$ and that our assertion holds for $k - 1$.

If n is the root, then, obviously, n' is not the root. Hence, $\mathsf{alp}(k, n) \prec \mathsf{alp}(k, n')$. Since $(n', m') \in L_{k,n',l}(D)$, by Proposition 27, it follows from Definition 32 that $(n', m') \notin \texttt{label}_{k,(n,m)}(D)$.

If n is not the root, but n' is the root, then $\mathsf{length}(\mathsf{alp}(k, n)) \geq 1$, and, by Definition 32, $(n', n') \notin L_{k,n,l}$. Consequently, $(n', m') \notin \texttt{label}_{k,(n,m)}(D)$.

We may therefore assume in the remainder of this proof that neither n nor n' is the root. Thus, let p be the parent of n and let p' be the parent of n'. We consider two cases.

(1) $l \geq 1$. Then, both (p, m) and (p', m') are in $\mathsf{UpPaths}(D, k-1)$. Since $(n, m) \not\equiv_{P(k)} (n', m')$, and n and n' have the same label, we know from Definition 7 that $(p, m) \not\equiv_{P(k-1)} (p', m')$. By the induction hypothesis, we have that $(p', m') \notin \texttt{label}_{k-1,(p,m)}(D)$. If $(p', m') \notin L_{k-1,p,l-1}(D)$, then $(n', m') \notin \ell;\uparrow;L_{k-1,p,l-1}(D) = L_{k,n,l}(D)$, whence $(n', m') \notin \texttt{label}_{k,(n,m)}(D)$. Otherwise, there exists a node p'' such that $\mathsf{alp}(k-1, p) \prec \mathsf{alp}(k-1, p'')$ and $(p', m') \in L_{k-1,p'',l-1}$. But this is only possible if $\mathsf{alp}(k-1, p'') \prec \mathsf{alp}(k-1, p')$. Hence, $\mathsf{alp}(k-1, p) \prec \mathsf{alp}(k-1, p')$. It follows that $\mathsf{alp}(k, m) \prec \mathsf{alp}(k, m')$. Now, by Proposition 27, $(p', m') \in L_{k-1,p',l-1}$, whence $(n', m') \in \ell \,;\uparrow\,; L_{k-1,p',l-1} = L_{k,n',l}$. Consequently, $(n', m') \notin \texttt{label}_{k,(n,m)}(D)$.

(2) $n = n'$ and $m = m'$. If $(p, p) \equiv_{P(k-1)} (p', p')$, it would follow from Definitions 7 and 5 that $(n, n) \equiv_{P(k)} (n', n')$, a contradiction. Hence, $(p, p) \not\equiv_{P(k-1)} (p', p')$. By the induction hypothesis, we have that $(p', p') \notin \texttt{label}_{k-1,(p,p)}(D)$. Assume first that $(p', p') \notin L_{k-1,p,0}(D)$. Then $(n', n') \notin \ell[\uparrow;L_{k-1,p,0}](D)$. It is readily seen that the $\mathscr{U}(k)$ expression $\ell[\uparrow;L_{k-1,p,0}](D)$ is equivalent to $L_{k,n,0}$, whence $(n', n') \notin L_{k,n,0}$. Consequently, $(n', n') \notin \texttt{label}_{k,(n,n)}(D)$. Otherwise, there exists a node p'' such that $\mathsf{alp}(k-1, p) \prec \mathsf{alp}(k-1, p'')$ and $(p', p') \in L_{k-1,p'',0}$. But this is only possible if $\mathsf{alp}(k-1, p'') \prec \mathsf{alp}(k-1, p')$. Hence, $\mathsf{alp}(k-1, p) \prec \mathsf{alp}(k-1, p')$. It follows that $\mathsf{alp}(k, m) \prec \mathsf{alp}(k, m')$. Now, by Proposition 27, $(p', p') \in L_{k-1,p',0}$. It follows that $(n', n') \in \ell[\uparrow\,;\,L_{k-1,p',0}](D)$. It is readily seen that then the $\mathscr{U}(k)$ expression $\ell[\uparrow\,;\,L_{k-1,p',0}](D)$ is equivalent to $L_{k,n',0}$, whence $(n', n') \in L_{k,n,0}$. Consequently, $(n', n') \notin \texttt{label}_{k,(n,n)}(D)$.

In other words, we have that for each block B, an expression $\texttt{label}_B \in \mathscr{U}(k)$ can be constructed such that $\texttt{label}_B(D) = B$, completing the proof of Proposition 34. These are precisely the labeling expressions of Theorem 25.

5 Towards indexes: $A(k)$-based, or $P(k)$-based?

In Section 3, we argued that many XPath queries can be evaluated by (1) discovering appropriate blocks of $P(k)$-partitions and (2) assembling these blocks, typically through unions and joins, into the final answer. Discovering appropriate blocks of the $P(k)$-partition was accomplished through decomposition and inversion techniques. Relative to a $P(k)$-partition, these techniques yield expressions in $\mathscr{D}(k)$ and $\mathscr{U}(k)$ without predicate operations. Through the Label-Union Theorem developed in Section 4, we know that these expressions can be associated with label expressions, which are syntactic objects that identify the relevant blocks. Thus, to develop an index structure to support these evaluations, we need a data structure that organizes these label expressions and their associated partition blocks in a way that allows fast look up. Given the simplicity of the labeling expressions, this is entirely feasible. One of the potential drawbacks of such an index structure is that it can be large: for a given k, its size is $O(k|V|)$ where V is the set of nodes of the document. However, we believe that, in practice, storing such indexes will only be necessary for small k values, and as such their size is nearly linear in the size of the document.

Of course, it is also possible to develop indexes that are based on the $A(k)$-partitions. In fact, the $A(k)$-index introduced by Kaushik et al. [13] is an example of this. This index has several very desirable properties: (1) its size is $O(|V|)$ and (2) for expressions in $\mathscr{U}(k)$ without predicates wherein exactly k "$\uparrow$" primitives occur, simple navigations through the index yield their results. However, it has also some significant limitations. Some of these were already briefly discussed in Section 2.8. We elaborate a little further on this section, here. For example, consider an expression without predicates in $\mathscr{U}(j)$, $j > k$, that utilizes j "$\uparrow$" primitives. Such an expression can be written in the form $E_1 \,;\, E_2$ where $E_1 \in \mathscr{U}(k)$ and $E_2 \in \mathscr{U}(j-k)$. Now the $A(k)$-index can determine the set of nodes that are the result of evaluating E_1 on the document. However now, starting from these nodes, E_2 is to be evaluated, and this can only be done by accessing and navigating the original document tree. (Notice that an index based on the $P(k)$-partitions does not suffer from this problem because it never requires extra navigation in the document.) A very similar problem occurs with expressions that have predicates. Consider an expression in $\mathscr{U}(j)$ of the form $E_1[E_2]$, where $E_1 \in \mathscr{U}(k)$ and $E_2 \in \mathscr{U}(j-k)$. Again, the $A(k)$-index can support E_1 well and retrieve the set of nodes that are the result of its evaluation. But again, to process the predicate $[E_2]$, it

is necessary to navigate the original document. Notice, again, that the $P(k)$-based indexes do not suffer from this problem.

From this discussion, we conclude that $P(k)$-based indexes are to be preferred over $A(k)$-based indexes, especially when only small k's are sufficient.

In fact, we are currently implementing $A(k)$- and $P(k)$-based index structures. Some preliminary results have already been obtained [2]. These experiments confirm that $P(k)$-indexes can outperform $A(k)$-indexes by several orders of magnitude, outweighing the relatively moderate overhead involved in their construction. Additionally, real data tend to be fairly shallow [16]. As a consequence, dramatic improvements in performance already occur for relatively small values of k. Our first results also indicate that, beyond the initial drop in query execution time, no significant improvements occur for larger values of k. We may therefore conclude that the joins required in some of the evaluation schemes outlined in the present paper have no significant adverse effect on the overall performance.

6 Future Directions

In this paper, we take a fresh step towards establishing connections between the theoretical study of query languages and engineering research on the design and implementation of XML database systems. These connections hinge on a new methodology for coupling *index*-induced partitions and *language*-induced partitions of an XML document.

To take full advantage of the $P(k)$-partitions introduced here and their block labeling expressions, we next need a data structure that is capable of locating all partition blocks based on label look-up, and in which the partition blocks that participate in the evaluation of a query are stored close to each other and can be located with a minimum number of label look-ups. We are currently involved in the development of a data structure which satisfies these requirements.

In addition, we are going to investigate how information on the query workload can drive various choices to be made, such as the appropriate value of k or the label paths to be indexed.

Other future work includes experiments with queries involving ancestor/descendant operations in combination with structural joins [1].

Acknowledgments. We thank the referees for their helpful comments.

References

[1] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, D. Srivastava, Structural Joins: A Primitive for Efficient XML Query Pattern Matching, in: IEEE ICDE, San Jose, California, 2002.

[2] S. Brenes, Y. Wu, D. V. Gucht, P. S. Cruz, Trie Indexes for Efficient XML Query Evaluation, in: WebDB, Vancouver, Canada, 2008.

[3] N. Bruno, N. Koudas, D. Srivastava, Holistic Twig Joins: Optimal XML Pattern Matching, in: ACM SIGMOD, Madison, Wisconsin, 2002.

[4] J. Clark, S. DeRose (eds.), XML Path Language (XPath) Version 1.0, `http://www.w3.org/TR/XPATH`.

[5] D. Comer, The Ubiquitous B-Tree, ACM Comput. Surv. 11 (2) (1979) 121–137.

[6] R. Goldman, J. Widom, DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases, in: VLDB, Athens, Greece, 1997.

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

[8] G. Gou, R. Chirkova, Efficiently Querying Large XML Data Repositories: A Survey, IEEE Trans. Knowledge and Data Eng. 19 (10) (2007) 1381–1403.

[9] M. Gyssens, J. Paredaens, D. Van Gucht, G. H. L. Fletcher, Structural Characterizations of the Semantics of XPath as Navigation Tool on a Document, in: ACM PODS, Chicago, 2006.

[10] H. He, J. Yang, Multiresolution Indexing of XML for Frequent Queries, in: IEEE ICDE, Boston, 2004.

[11] R. Kaushik, P. Bohannon, J. F. Naughton, H. F. Korth, Covering Indexes for Branching Path Queries, in: ACM SIGMOD, Madison, Wisconsin, 2002.

[12] R. Kaushik, R. Krishnamurthy, J. F. Naughton, R. Ramakrishnan, On the Integration of Structure Indexes and Inverted Lists, in: ACM SIGMOD, Paris, France, 2004.

[13] R. Kaushik, P. Shenoy, P. Bohannon, E. Gudes, Exploiting Local Similarity for Indexing Paths in Graph-Structured Data, in: IEEE ICDE, San Jose, CA, 2002.

[14] C. Koch, Processing Queries on Tree-Structured Data Efficiently, in: ACM PODS, Chicago, 2006.

[15] T. Milo, D. Suciu, Index Structures for Path Expressions, in: ICDT, Jerusalem, 1999.

[16] I. Mlynkova, K. Toman, J. Pokorný, Statistical Analysis of Real XML Data Collections, in: COMAD, New Delhi, India, 2006.

[17] M. M. Moro, Z. Vagena, V. J. Tsotras, Tree-Pattern Queries on a Lightweight XML Processor, in: VLDB, Trondheim, Norway, 2005.

[18] C. Qun, A. Lim, K. W. Ong, D(k)-Index: An Adaptive Structural Summary for Graph-Structured Data, in: ACM SIGMOD, San Diego, California, 2003.

[19] P. Ramanan, Covering Indexes for XML Queries: Bisimulation – Simulation = Negation, in: VLDB, Berlin, 2003.

[20] K. Runapongsa, J. M. Patel, R. Bordawekar, S. Padmanabhan, XIST: An XML Index Selection Tool, in: XSym, Toronto, 2004.

[21] D. Sangiorgi, On the Origins of Bisimulation and Coinduction, ACM Trans. Program. Lang. Syst., to appear.

[22] K. Yi, H. He, I. Stanoi, J. Yang, Incremental Maintenence of XML Structural Indexes, in: ACM SIGMOD, Paris, 2004.

[23] C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, G. M. Lohman, On Supporting Containment Queries in Relational Database Management Systems, in: ACM SIGMOD, Santa Barbara, California, 2001.