An Algebraic Query Model for Effective and Efficient Retrieval of XML Fragments

Sujeet Pradhan
Kurashiki University of Science and the Arts
Nishinoura 2640, Tsurajima-cho
Kurashiki, Japan
sujeet@cs.kusa.ac.jp

ABSTRACT

Finding a suitable fragment of interest in a non-schematic XML document with a simple keyword search is a complex task. To deal with this problem, this paper proposes a theoretical framework with a focus on an algebraic query model having a novel query semantics. Based on this semantics, XML fragments that look meaningful to a keyword-based query are effectively retrieved by the operations defined in the model. In contrast to earlier work, our model supports filters for restricting the size of a query result, which otherwise may contain a large number of potentially irrelevant fragments. We introduce a class of filters having a special property that enables significant reduction in query processing cost. Many practically useful filters fall in this class and hence, the proposed model can be efficiently applied to real-world XML documents. Several other issues regarding algebraic manipulation of the operations defined in our query model are also formally discussed.

1. INTRODUCTION AND MOTIVATION

While it is widely-accepted that keyword search is the most friendly interface for querying XML documents, there is considerably less agreement on how to answer such keyword-based queries. Given a set of keywords as a query, there is no consensus on what portion or portions of an XML document should be retrieved as *the* answer.

An XML document is commonly modelled as a rooted tree (either ordered or unordered). The problem of finding a portion of interest in an XML document is thus equivalent to the problem of identifying an appropriate subtree of the tree which represents the document in consideration. While several studies have been done in the recent past regarding this issue in the literature[4][5][12][15][20], the primary focus has been on so-called *data-centric* XML documents such as bibliographic data. Data-centric XML documents are highly schematic and their element tag names are generally "semantically meaningful". As a result, one can exploit both the schema and the tag names to identify meaningful XML fragments to some precision. For example in [4] and [5], document components are indexed according to their interrelationships which are determined by analysing semantic meanings of tag names such as < `book` >, < `author` > in a document.

In contrast, a *document-centric* XML document such as the one shown in Figure 1, hardly has any fixed schema, usually has long textual contents, and typically has tag names such as < `section` >, < `subsection` >, < `par` > etc. which only describe structural relationship, but offer little help in determining any semantic relationship among document components.

It is often argued that given a set of keywords as a query against an XML tree, the smallest subtree containing all the keywords is enough to answer this query[5][7][12][15][20]. While this argument seems logical enough in the realm of data-centric XML documents, it is not guaranteed to be effective to compute even a simple and intuitive answer to a query against general document-centric XML documents. As an illustration, consider a query {`XQuery`, `optimization`} against an XML document shown in Figure 1. According to the conventional query semantics, the smallest subtree containing both the keywords `XQuery` and `optimization` (the paragraph represented by node **n17** in Figure 1) would be the answer to this query. However, a general user may find the fragment represented by the nodes **n16**, **n17**, and **n18** more intuitive and more appropriate since it is self-contained and more informative than **n17** alone. Our first challenge is how to retrieve such a fragment as *one single answer unit* effectively by merely exploiting the structural relationship among the components of the underlying data.

Obviously the problem of retrieval unit in a document-centric XML document is more complex than in a data-centric one mainly because of the facts that 1) document-centric XML documents are non-schematic; 2) the tag names are not guaranteed to carry any literal semantics that would assist in determining the retrieval unit and 3) there is no prior knowledge of how keywords would be split across the nodes of a desired XML subtree (refer to Figure 2). As a result, we need a more effective query mechanism, which goes beyond the smallest subtree semantics, for identifying potential fragments of interest, particularly in document-centric XML documents.

In this paper, we intend to compute fragments of interest, which in most cases, may be larger than the fragments that would have been retrieved according to the smallest subtree semantics. One question that naturally arises then is: How large a fragment is large enough? There is no con-

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.
VLDB '06, September 12-15, 2006, Seoul, Korea.
Copyright 2006 VLDB Endowment, ACM 1-59593-385-9/06/09.

```
<proceedings title="VLDB Conference"> n0
  <paper title="A New Query Language for XML data"> n1
    <section heading="Introduction"> n2
      <par> ... XML data is ...</par> n3
    </section>
    ...
    <section heading="Query Processing"> n14
      <par>.. logical ... ... ... </par> n15
      <subsection heading="Optimization Issues"> n16
        <par> ... successful implementation of XQuery ... query optimization ... </par> n17
        <par> ... some of these techniques in XQuery are borrowed from ... </par> n18
      </subsection>
    </section>
    ...
  </paper>
  ...
  <paper title="Efficient Stream Data Management"> n79
    <section heading="Introduction and Motivation"> n80
      <par> ... optimization is an integral part of any ... </par> n81
    </section>
    ...
  </paper>
  ...
</proceedings>
```

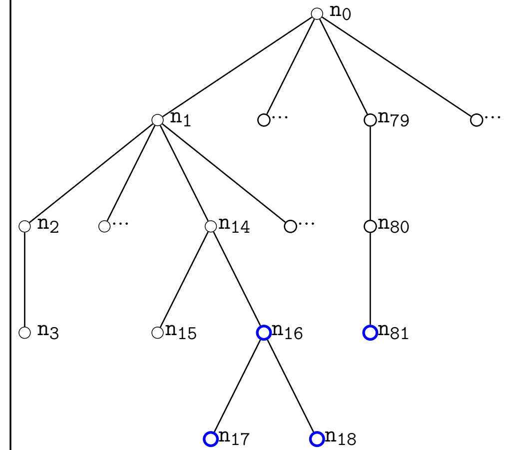

Figure 1: An Example XML Document and its Tree Representation

crete answer to this question as the old adage goes, "One size does not fit all". Although no one fixed size (or shape) of a fragment meets all queries, this paper attempts to find relevant-looking fragments; a relevant-looking fragment in this paper means a subtree containing all the query keywords, and, in addition, this subtree would have no extraneous nodes. Having said that, however, we do not wish to generate each and every computable fragment, because this may return an answer set with several potentially irrelevant fragments. For example, consider the fragment represented by the nodes **n0**,**n1**,**n14**,**n16**,**n17**,**n18**,**n79**,**n80**,**n81** in Figure 1. Even though this fragment consists of the nodes containing all the query keywords in our example query {`XQuery`, `optimization`}, it may still be considered irrelevant because a substantial portion of its contents are unrelated to each other. Not only will this overwhelm the user with a huge number of often irrelevant fragments, but it will also degrade the performance of the overall system. Therefore, our next challenge is how to avoid computation of fragments that are potentially irrelevant to a user.

This paper describes the formal framework of an algebraic query model designed to address the two challenges stated above for an effective and efficient XML keyword search. This query model is supported by *selection* and *join* operations; the two primary operations in traditional database systems. Keyword queries are transformed into algebraic expressions and XML fragments of interest are effectively computed by joining several document components whose contents would contain the specified keywords.

We then shift our focus on restricting the generation of a large number of irrelevant fragments in an answer set. For that purpose, we provide various filters, which in fact are selection predicates. Our main interest is however, not only in the restriction of the size of the query result, but also in investigating whether such filters would contribute to significant reduction of query processing cost. We explore the algebraic properties of several filters and show that not all filters are capable of contributing to query optimization. In addition, this paper proves that *selection* with a specific class of filters, what we call *anti-monotonic* filters, can indeed be pushed down in the query evaluation tree ahead of *join* operations. As a result, by incorporating practically useful filters with anti-monotonic property in our query mechanism, we can expect a substantial performance gain in our query processing. We also discuss several other algebraic manipulation of the operations of our model, which could enable us to achieve better processing efficiency under certain conditions.

Our contributions in this paper can be summarized as:

- To find appropriate fragments of interest with a simple keyword search in a non-schematic XML document, we provide a novel query semantics — a semantics that is different from the smallest subtree semantics commonly adopted for schematic XML documents.

- Our algebra has a strong theoretical foundation with several promises for query optimization, which can be exploited by a query processor for performance gain.

- To prevent an overwhelming size of an answer set, we propose database style filtering instead, which would complement existing XML retrieval systems that apply IR-style ranking techniques.

- Although no experiments have been conducted yet to verify the viability of our model, we provide formal proofs and convincing examples to justify our claims.

The rest of the paper is organized as follows. In Section 2, query model is formalized by defining all the algebraic operations required to compute answers to a query. Optimization techniques are discussed in Section 3. In Section 4, we give an illustrative example to explain different query evaluation strategies that the model offers. Section 5 sheds some light on a few important issues that are not fully investigated in this paper. Related work is provided in Section 6, and finally we conclude by highlighting our contributions.

Figure 2: A few possible variations of the way two keywords k_1 and k_2 are split across the nodes of the target subtrees of interest

2. QUERY MODEL

In this section, we formally describe our query model. Preliminary ideas regarding this model are also given in our earlier work[14].

2.1 Basic Definitions

Definition 1 (Document). *An XML document, is a rooted ordered tree $\mathcal{D} = (\mathtt{N}, \mathtt{E})$ with a set of nodes* N *and a set of edges $\mathtt{E} \subseteq \mathtt{N} \times \mathtt{N}$. There exists a distinguished root node from which the rest of the nodes can be reached by traversing the edges in* E*. Each node except the root has a unique parent node.*

Each node n of the document tree is associated with a logical component such as < **section** > or < **par** > of the document. As in [7][9], we do not distinguish between tag/attribute names and text contents. There is a function keywords(n) that returns the representative keywords of the corresponding component in n. The nodes are arranged in such a way that the depth-first pre-order traversal of the tree would preserve the topology of the document. We write $\mathtt{nodes}(\mathcal{D})$ for all the nodes N.

Definition 2 (Document Fragment). *Let $\mathcal{D}$ be an XML document. Then $\mathtt{f} \subseteq \mathcal{D}$ is a document fragment, or simply a fragment, iff $\mathtt{nodes}(\mathtt{f}) \subseteq \mathtt{nodes}(\mathcal{D})$ and the subgraph induced by* nodes(f) *in $\mathcal{D}$ is a rooted tree. In other words, the induced subgraph is connected.*

A fragment can thus be denoted by a subset of nodes in a document tree — the tree induced by which is also a rooted ordered tree. A fragment may consist of only a single node or all the nodes which constitute the whole document tree. In Figure 1, the set of nodes $\langle \mathtt{n16}, \mathtt{n17}, \mathtt{n18} \rangle$[1] is a fragment of the sample document tree. Hereafter, unless stated otherwise, the first node of a fragment represents the root of the tree induced by it. For clarity, we refer to a single-node fragment simply as a node.

2.2 Algebra

To formally define the operational semantics of a query, we first need to define operations on fragments and sets of fragments. The operations can be basically classified as (1) *selection* and (2) *join* operations.

Definition 3 (Selection). *Supposing* F *be a set of fragments of a given document, and* P *be a predicate which maps a document fragment into true or $false$, a* selection *from* F *by the predicate* P*, denoted by $\sigma_{\mathtt{P}}$, is defined as a subset $\mathtt{F}'$ of* F *such that $\mathtt{F}'$ includes all and only fragments satisfying* P*. Formally, $\sigma_{\mathtt{P}}(\mathtt{F}) = \{\mathtt{f} \mid \mathtt{f} \in \mathtt{F}, \mathtt{P}(\mathtt{f}) = true\}$.*

[1]For clarity, we use '⟨' and '⟩' instead of conventional '{' and '}' to enclose the nodes of a fragment

Hereafter, the predicate P is also called a filter of the *selection* $\sigma_{\mathtt{P}}$.

The simplest filter is for the keyword selection of the type 'keyword $= k$' which selects only those document fragments having the word 'k'. Several other filters will be introduced in Section 3.3 below.

Next, we define various *join* operations on document fragments.

Definition 4 (Fragment Join). *Let $\mathtt{f}_1, \mathtt{f}_2, \mathtt{f}$ be fragments of the document tree $\mathcal{D}$. Then, fragment join between $\mathtt{f}_1$ and $\mathtt{f}_2$ denoted by $\mathtt{f}_1 \bowtie \mathtt{f}_2$ is* f *iff*

1. $\mathtt{f}_1 \subseteq \mathtt{f}$,
2. $\mathtt{f}_2 \subseteq \mathtt{f}$ *and*
3. $\nexists \mathtt{f}'$ *such that* $\mathtt{f}' \subseteq \mathtt{f} \wedge \mathtt{f}_1 \subseteq \mathtt{f}' \wedge \mathtt{f}_2 \subseteq \mathtt{f}'$

Intuitively, the fragment join operation takes two fragments $\mathtt{f}_1$ and $\mathtt{f}_2$ of $\mathcal{D}$ as its input and finds the minimal fragment f in $\mathcal{D}$ such that the resulting fragment would contain both the input fragments $\mathtt{f}_1$ and $\mathtt{f}_2$, and there exists no other smaller fragment $\mathtt{f}'$ contained by f in $\mathcal{D}$, which would also contain the input fragments $\mathtt{f}_1$ and $\mathtt{f}_2$. Figure 3 (b) shows the operation between two fragments $\langle \mathtt{n4}, \mathtt{n5} \rangle$ and $\langle \mathtt{n7}, \mathtt{n9} \rangle$ (refer to Figure 3 (a)) which finds its minimal subgraph fragment $\langle \mathtt{n3}, \mathtt{n4}, \mathtt{n5}, \mathtt{n6}, \mathtt{n7}, \mathtt{n9} \rangle$ (fragment inside dashed line in Figure 3 (b)). By its definition, the fragment join operation between arbitrary fragments $\mathtt{f}_1$, $\mathtt{f}_2$ and $\mathtt{f}_3$ has the following algebraic properties.

Idempotency $\mathtt{f}_1 \bowtie \mathtt{f}_1 = \mathtt{f}_1$
Commutativity $\mathtt{f}_1 \bowtie \mathtt{f}_2 = \mathtt{f}_2 \bowtie \mathtt{f}_1$
Associativity $(\mathtt{f}_1 \bowtie \mathtt{f}_2) \bowtie \mathtt{f}_3 = \mathtt{f}_1 \bowtie (\mathtt{f}_2 \bowtie \mathtt{f}_3)$
Absorption $\mathtt{f}_1 \bowtie (\mathtt{f}_2 \subseteq \mathtt{f}_1) = \mathtt{f}_1$

These properties not only enable an easy implementation of the operations but also lay foundation for optimizing query evaluation by enabling algebraic manipulation of operations defined further below.

Next, we extend this operation to a set of fragments. called *pairwise fragment join*, which is the set-variant of fragment join.

Definition 5 (Pairwise Fragment Join). *Let $\mathtt{F}_1$ and $\mathtt{F}_2$ be two sets of fragments in a document $\mathcal{D}$,* pairwise fragment join *of $\mathtt{F}_1$ and $\mathtt{F}_2$, denoted by $\mathtt{F}_1 \bowtie \mathtt{F}_2$, is defined as a set of fragments yielded by taking* fragment join *of every combination of an element in $\mathtt{F}_1$ and an element in $\mathtt{F}_2$ in a pairwise manner. Formally,*

$$\mathtt{F}_1 \bowtie \mathtt{F}_2 = \{\mathtt{f}_1 \bowtie \mathtt{f}_2 \mid \mathtt{f}_1 \in \mathtt{F}_1,\ \mathtt{f}_2 \in \mathtt{F}_2\}.$$

Figure 3 (a),(c) illustrates an example of *pairwise fragment join* operation. For given $\mathtt{F}_1 = \{\mathtt{f}_{11}, \mathtt{f}_{12}\}$ and $\mathtt{F}_2 = \{\mathtt{f}_{21}, \mathtt{f}_{22}\}$, $\mathtt{F}_1 \bowtie \mathtt{F}_2$ produces a set of fragments $\{\mathtt{f}_{11} \bowtie \mathtt{f}_{21}, \mathtt{f}_{11} \bowtie \mathtt{f}_{22}, \mathtt{f}_{12} \bowtie \mathtt{f}_{21}, \mathtt{f}_{12} \bowtie \mathtt{f}_{22}\}$.

For arbitrary fragment sets $\mathtt{F}_1$, $\mathtt{F}_2$, and $\mathtt{F}_3$, *pairwise fragment join* has the following algebraic properties.

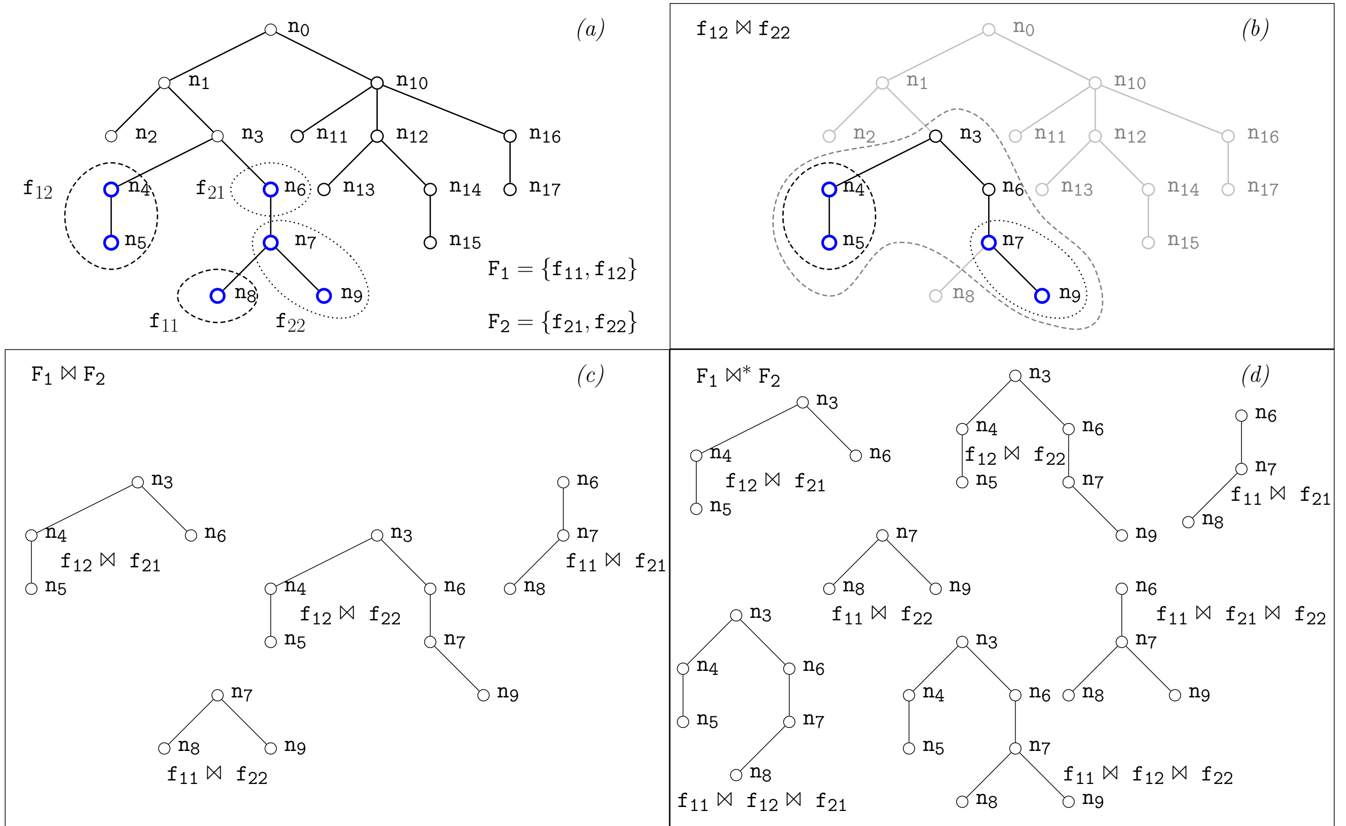

Figure 3: (a) A Document Tree (b) Fragment Join (c) Pairwise Fragment Join and (d) Powerset Fragment Join Operations

Commutativity $F_1 \bowtie F_2 = F_2 \bowtie F_1$

Associativity $(F_1 \bowtie F_2) \bowtie F_3 = F_1 \bowtie (F_2 \bowtie F_3)$

Monotonicity $F_1 \bowtie F_1 \supseteq F_1$

Distributive Law $F_1 \bowtie (F_2 \cup F_3) = (F_1 \bowtie F_2) \cup (F_1 \bowtie F_3)$

The *pairwise fragment join* operation does not satisfy the idempotency property as we can easily prove by showing counter examples for it.

We now define *powerset fragment join* — another variant of the *fragment join* operation.

Definition 6 (Powerset Fragment Join). *Let F_1 and F_2 be two sets of fragments in a document $\mathcal{D}$,* powerset fragment join *between F_1 and F_2, denoted by $F_1 \bowtie^* F_2$, is defined as a set of fragments produced by applying* fragment join *operation to an arbitrary number (but not 0) of elements in F_1 and F_2. Formally,*

$$F_1 \bowtie^* F_2 = \{\bowtie (F'_1 \cup F'_2) \mid F'_1 \subseteq F_1, F'_2 \subseteq F_2, F'_1 \neq \phi, F'_2 \neq \phi\}$$
$$\text{where } \bowtie \{f_1, f_2, \ldots, f_n\} = f_1 \bowtie \ldots \bowtie f_n.$$

Figure 3 (a),(d) illustrates an example of *powerset fragment join* operation. It should be noted here that for the same two sets of fragments $F_1 = \{f_{11}, f_{12}\}$ and $F_2 = \{f_{21}, f_{22}\}$ in Figure 3 (a), *powerset fragment join* produces more fragments than *pairwise fragment join* (refer to Figure 3 (c)). It should also be noted that some of the fragments are produced more than once due to the algebraic properties of *fragment join* and *pairwise fragment join*.

2.3 Query Evaluation

Definition 7 (Query). *A query can be denoted by $Q_P\{k_1, k_2, ..., k_m\}$ where k_j is called a query term for all $j = 1, 2, \ldots, m$ and P is a selection predicate.*

We write $k \in \mathtt{keywords}(n)$ to denote that query term k appears in the textual contents (element contents in XML terminology) associated with the node n.

Definition 8 (Query Answer). *Given a query $Q_P\{k_1, k_2, ..., k_m\}$, answer A to this query is a set of document fragments defined to be*

$$\{\ f \mid (\forall k \in Q) \exists n \in f : n \text{ is a leaf node of } f \ \wedge \ k \in \mathtt{keywords}(n) \wedge P(f) = \mathit{true}\ \}.$$

Note that as in [7], we also adopt conjunctive query semantics. Intuitively, an answer to a query is a document fragment consisting of several structurally-related logical components and each keyword in the query must appear in at least one component that constitutes the fragment. In addition, the fragment must satisfy the selection predicate(s) specified in the query.

A query represented by $\{k_1, k_2\}$ and a selection predicate P against a document $\mathcal{D}$ can be evaluated by the following formula.

$$Q_P\{k_1, k_2\} \quad = \quad \sigma_P(F_1 \bowtie^* F_2)$$

where $F_1 = \sigma_{\mathsf{keyword}=k_1}(F)$, $F_2 = \sigma_{,\mathsf{keyword}=k_2}(F)$ and $F = \mathtt{nodes}(\mathcal{D})$.

Since *powerset fragment join* operation often can be very expensive, evaluation of an answer set as it is may turn out very costly (See Section 4 for an example of query evaluation). In the following section, we provide several optimization techniques for reducing the cost of answer evaluation.

3. OPTIMIZATION ISSUES

Query processors in a traditional database management system have two basic kinds of optimizations: algebraic manipulation and cost-estimation strategies[18]. The focus of this paper would be on the former.

Given an algebraic expression, the main goal of algebraic manipulation is to investigate if there exist other equivalent, but more efficient, algebraic expressions. If such alternative expressions are available, then a query processor can derive logically optimized query plans in order to minimize the response time. Note that such plans can be derived irrespective of how they are implemented.

In this section, we present several algebraic transformation rules crucial for optimization of our query model. First, we provide an equivalent expression for the *powerset fragment join* operation, which otherwise is an expensive operation. Second, we discuss how selection operations can be pushed down in a query evaluation tree, if the selection predicates belong to a class of filters having a simple property.

3.1 Algebraic Manipulation of Powerset Fragment Join

The implementability of *powerset fragment join* operation comes in question, especially when the input size is very large. However, we show that *powerset fragment join* can be transformed into an equivalent expression whose computation cost, under certain circumstances, can be much less than the original expression. We first define several new operations required for this equivalent expression.

DEFINITION 9 (FIXED POINT). *If* $\mathtt{F}$ *be a set of fragments of the document tree* $\mathcal{D}$*, its* fixed point $\mathtt{F}^+$ *is defined as*

$$\mathtt{F}^+ = \{\bowtie \mathtt{F_i} \mid \mathtt{F_i} \subseteq \mathtt{F} \wedge \mathtt{F_i} \neq \emptyset\}.$$

Intuitively, given a fragment set, the *fixed point* of a fragment set is another fragment set that would include all the fragments obtained by performing *fragment join* operation on every possible combination of an arbitrary number of fragments of the given set. A naive algorithm can compute *fixed point* of a fragment set by performing *pairwise fragment join* operation $2^{|\mathtt{F}|}$ times on itself where $|\mathtt{F}|$ is the number of fragments in the set. Obviously, such an algorithm is impractical for a large value of $|\mathtt{F}|$. Below, we provide several solutions to overcome this problem.

3.1.1 Naive Solution

An obvious solution to this problem is to devise an algorithm using dynamic programming technique. This is possible because the definition of *fixed point* given above can be expanded as:

$$\mathtt{F}^+ = \mathtt{F} \cup (\mathtt{F} \bowtie \mathtt{F}) \cup (\mathtt{F} \bowtie \mathtt{F} \bowtie \mathtt{F}) \cup \ldots$$

Computation of *fixed point* can then be done by performing *pairwise fragment join* operation iteratively between a pair of the given set and the intermediate set generated in the previous iteration until the resulting set stabilizes. At this point, no new fragments are generated by any further *pairwise fragment join* operation and the computation can be stopped. However, evaluating *fixed point* even with this technique may lead to redundant computation. This is because, after each iteration one must perform *fixed point checking* (that is checking whether a fragment set has reached its fixed point or not). As it will be explained later, fixed point computation is an integral part of our query processing and, among many other critical issues, its efficient evaluation is also crucial for the efficiency of our overall query mechanism. What follows here is a means to avoid overhead caused by *fixed point checking* computation during the evaluation of *fixed point* of a fragment set.

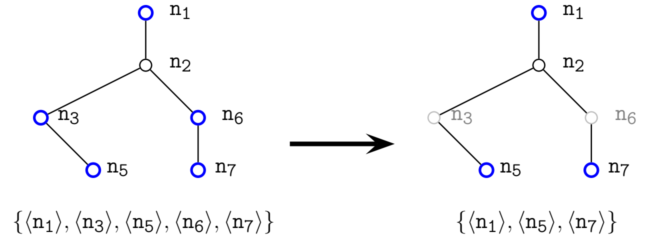

Figure 4: Fragment Set Reduction

3.1.2 Alternative Solution

The basic idea here is to derive the number of iterations required for obtaining the *fixed point* of a given fragment set before the actual fixed point computation. This would eliminate the need of *fixed point checking* and consequently speed up the convergence to the *fixed point* of a fragment set. Interestingly, we have observed that the number of iterations required to obtain the *fixed point* of a fragment set depends not on the total number of elements in the set, but on the number of elements in one of its subsets. This subset is unique and would not contain any fragment which would be subsumed by a resulting fragment produced by *fragment join* operation between any two arbitrary elements of the set. Based on this observation, we define an operation that would reduce a fragment set to a smaller set by eliminating all those fragments, which are redundant for estimating the required number of iterations.

DEFINITION 10 (FRAGMENT SET REDUCE). *Let* $\mathtt{F}$ *be a fragment set. Then the* fragment set reduce *operation on* $\mathtt{F}$ *is defined as*

$$\ominus(\mathtt{F}) = \{\mathtt{f} \mid \nexists \mathtt{f}', \mathtt{f}'' \in \mathtt{F} \text{ such that } \mathtt{f} \subseteq \mathtt{f}' \bowtie \mathtt{f}''\}$$

where $\mathtt{f}, \mathtt{f}', \mathtt{f}''$ *are all distinct fragments in* $\mathtt{F}$.

We call $\ominus(\mathtt{F})$ the *reduced set* of $\mathtt{F}$. Intuitively, given a fragment set $\mathtt{F}$, *fragment set reduce* operation eliminates those fragments from $\mathtt{F}$, which are the sub-fragments of fragments obtained from any two arbitrary fragments in the set $\mathtt{F}$.

THEOREM 1. *The cardinality of the* reduced set *denoted by* $|\ominus(\mathtt{F})|$ *gives the number of iterations required to obtain the* fixed point *of the fragment set* $\mathtt{F}$. *That is, if* $|\mathtt{F}|$ *is* n *and* $|\ominus(\mathtt{F})|$ *is* k $(k <= n)$*, and supposing* $\bowtie_n(\mathtt{F})$ *is a*

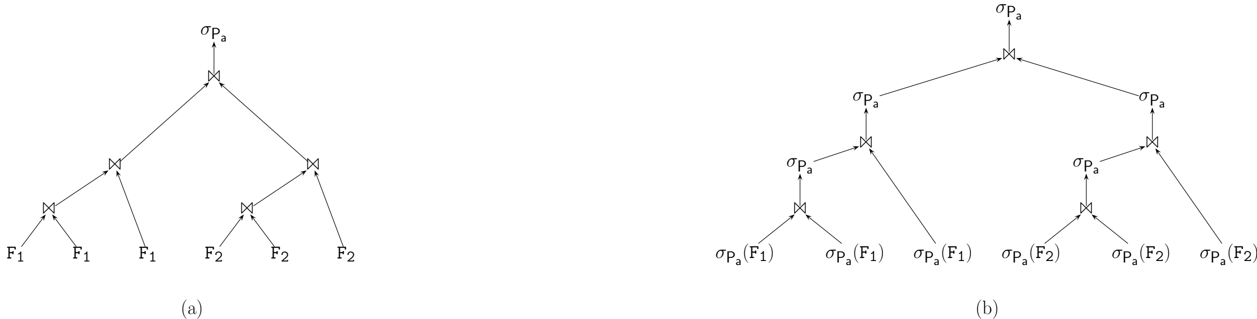

Figure 5: (a) Initial Query Evaluation Tree for $\sigma_{P_a}((F_1 \bowtie F_1 \bowtie F_1) \bowtie (F_2 \bowtie F_2 \bowtie F_2))$ (b) Equivalent Query Evaluation Tree Implementing 'Push-down' Strategy

short form to denote the fragment set obtained by performing pairwise fragment join *($\bowtie$) operation on n number of* F, *then* $\bowtie_n$ (F) $=\bowtie_k$ (F).

PROOF. *See Appendix.* □

In Figure 4, a set of fragments $F = \{\langle n_1\rangle, \langle n_3\rangle, \langle n_5\rangle, \langle n_6\rangle, \langle n_7\rangle\}$ is reduced to $\ominus(F) = \{\langle n_1\rangle, \langle n_5\rangle, \langle n_7\rangle\}$ after eliminating $\langle n_3\rangle$ and $\langle n_6\rangle$ because they are the sub-fragments of $\langle n_1\rangle \bowtie \langle n_5\rangle$ and $\langle n_1\rangle \bowtie \langle n_7\rangle$ respectively. Here, since the cardinality of the *reduced set* is 3, $((F \bowtie F) \bowtie F)$ should give the *fixed point* F^+ of the fragment set F.

3.1.3 Transformation of Powerset Fragment Join

Having defined these two new operations, we are now ready to provide the algebraic transformation of *powerset fragment join* operation.

THEOREM 2. *The* powerset fragment join *operation between two fragment sets* F_1 *and* F_2 *in a document* $\mathcal{D}$ *can be transformed into the following equivalent expression:*

$$F_1 \bowtie^* F_2 = F_1^+ \bowtie F_2^+$$

where F_1^+ *and* F_2^+ *are fixed points of* F_1 *and* F_2 *respectively.*

PROOF. *Formal proof is omitted for space reasons.* □

We give an informal justification for this claim. The semantics of *powerset fragment join* between fragments sets F_1 and F_2 is to produce a fragment set F consisting of fragments generated by taking at least one fragment from each operand F_1 and F_2. We know that the fixed points of F_1 and F_2 consist of fragments produced by performing *fragment join* operation on each possible combination of element fragments in F_1 and F_2 respectively. Therefore, the *pairwise fragment join* operation between these two fixed points of F_1 and F_2 would produce the same resulting fragment set as the one that would have been produced by the *powerset fragment join* operation.

3.1.4 Significance of Fragment Set Reduction

Recall that F_1^+ can be obtained by performing $|\ominus(F_1)|$ number of $\bowtie$ on F_1 and similarly F_2^+ can be obtained by performing $|\ominus(F_2)|$ number of $\bowtie$ on F_2^+. In addition, we need to perform *fragment set reduce* operation on each F_1 and F_2 in order to obtain their respective $|\ominus(F_1)|$ and $|\ominus(F_2)|$. One obvious doubt that may come up in a reader's mind is: How significant the operation *fragment set reduce* really is? In other words, is it judicious to perform this operation under any circumstances? The simple answer is "no", although it may be fair to say it depends largely upon how these operations are going to be evaluated in the implementation level. In order to give a more definitive answer, we require a formal cost model that enables us to estimate cost of these operations. Cost models are beyond the scope of this paper. At this point, our emphasis is on discovering equivalent expressions for our algebraic operations, which can be exploited by a query processor for optimization opportunities. We shall elaborate our views on the necessity of a cost model in Section 5.

Intuitively, the amount of reduction in query processing cost due to *fragment set reduce* operation depends upon how significantly the fragment sets can be reduced. The larger the factor, by which the fragment sets are reduced, the more opportunities there are for optimization. In the worst case, if the original sets cannot be reduced at all, the overall computation cost of generating *all* potential fragments will remain considerably large. Consequently, it may still be impractical to generate *all* computable fragment first, and then, only afterwards, disregard the irrelevant ones. Below, we show how this problem can be overcome by introducing filters that fall under a special class.

3.2 Commuting Selection with Join Operations

One of the main principles for algebraic manipulation in conventional database systems is to perform *selection* as early as possible[18]. Our goal here is to apply the same principle to our query mechanism so that without affecting the end result, we can eliminate as many unnecessary fragments as possible at an early stage of query processing. However, in order to achieve this goal, we must ensure that *selection* can indeed be pushed down in the query evaluation tree; that is even if we perform *selection* ahead of *join*, we are still guaranteed the same desired result. Unfortunately, as it will be shown later, this commutativity between *selection* and *join* cannot be achieved for all types of filters. Only a class of filters having anti-monotonic property allows selection operation to be pushed down in the query tree.

Below we shall formally define an anti-monotonic filter and show how optimization can be achieved by 'push-down' strategy in cases when selection predicates have anti-monotonic property. Several filters having this property will also be introduced in the following subsections.

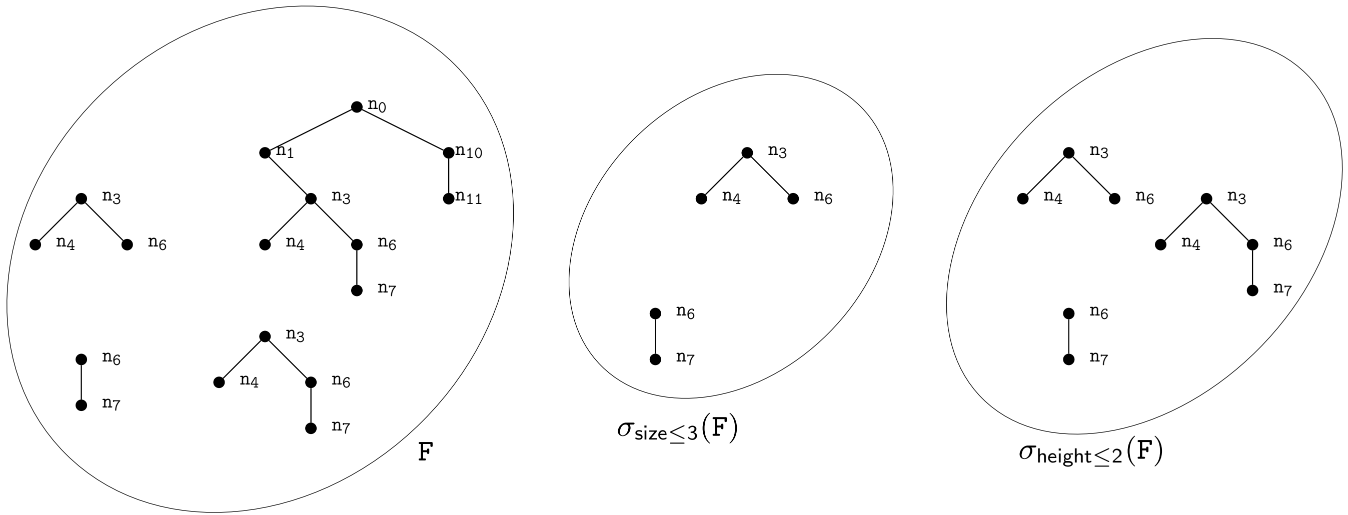

Figure 6: Anti-monotonic Filters

3.3 Anti-monotonic Filters

DEFINITION 11. *Given a fragment* f*, a filter* P *is anti-monotonic iff*

$$\forall \mathtt{f}' \subseteq \mathtt{f} : \mathsf{P}(\mathtt{f}) = \mathit{true} \Rightarrow \mathsf{P}(\mathtt{f}') = \mathit{true}$$

Thus, if a fragment f satisfies a filter predicate P, then all sub-fragments of f also satisfies P. In other words, if a filter P has anti-monotonic property, then for any fragment not satisfying P, none of its super-fragment can satisfy P either. It is obvious that both conjunction and disjunction of anti-monotonic filters have also anti-monotonic property. That is, if P_1 and P_2 are two distinct anti-monotonic filters, then $\mathsf{P}_1 \wedge \mathsf{P}_2$ and $\mathsf{P}_1 \vee \mathsf{P}_2$ are also anti-monotonic filters. However, the negation of an anti-monotonic filter does not retain the anti-monotonic property, as we can easily prove this by giving a counter example. We exclude negation of anti-monotonic filters from our further discussion. We denote a filter having anti-monotonic property as P_a whenever necessary to distinguish it from other filters.

THEOREM 3. Selection *with anti-monotonic filters can be pushed ahead of* pairwise fragment join *operation, that is,*

$$\sigma_{\mathsf{P}_\mathsf{a}}(\mathtt{F}_1 \bowtie \mathtt{F}_2) = \sigma_{\mathsf{P}_\mathsf{a}}(\sigma_{\mathsf{P}_\mathsf{a}}(\mathtt{F}_1) \bowtie \sigma_{\mathsf{P}_\mathsf{a}}(\mathtt{F}_2))$$

where $\mathtt{F}_1$ *and* $\mathtt{F}_2$ *are two (not necessarily distinct) fragment sets and* P_a *is an anti-monotonic filter.*

PROOF. *See Appendix.* □

From Theorem 3 we can conclude that *selection* with anti-monotonic filters can be pushed ahead of all *join* operations (see Figure 5). Therefore, in the presence of anti-monotonic filters, the *powerset fragment join* can be evaluated as:

$$\begin{aligned} \sigma_{\mathsf{P}_\mathsf{a}}(\mathtt{F}_1^+ \bowtie \mathtt{F}_2^+) &= \sigma_{\mathsf{P}_\mathsf{a}}(\sigma_{\mathsf{P}_\mathsf{a}}(\mathtt{F}_1^+) \bowtie \sigma_{\mathsf{P}_\mathsf{a}}(\mathtt{F}_2^+)) \\ &= \sigma_{\mathsf{P}_\mathsf{a}}(\sigma_{\mathsf{P}_\mathsf{a}}(\sigma_{\mathsf{P}_\mathsf{a}}(\mathtt{F}_1) \bowtie \sigma_{\mathsf{P}_\mathsf{a}}(\mathtt{F}_1) \bowtie \ldots \\ &\qquad \bowtie \sigma_{\mathsf{P}_\mathsf{a}}(\mathtt{F}_1)) \\ &\qquad \bowtie \sigma_{\mathsf{P}_\mathsf{a}}(\sigma_{\mathsf{P}_\mathsf{a}}(\mathtt{F}_2) \bowtie \sigma_{\mathsf{P}_\mathsf{a}}(\mathtt{F}_2) \bowtie \ldots \\ &\qquad \bowtie \sigma_{\mathsf{P}_\mathsf{a}}(\mathtt{F}_2))) \end{aligned}$$

Since both conjunction and disjunction of anti-monotonic filters also possess the anti-monotonic property, construction of more complex anti-monotonic filters is possible. Consequently, we can expect a significant performance gain if practically useful filters having anti-monotonic property *do* actually exist. The following subsection introduces several such filters.

3.3.1 Size of a Fragment

From a user's point of view, one natural way of restricting a query result is by specifying the size limit of a fragment. In other words, if the number of nodes of a answer fragment exceeds a certain value, it should be eliminated from the answer set. Beyond a certain size, the larger the fragment, the more possibility there is for that fragment to be irrelevant to the query. Supposing $\mathsf{size}(\mathtt{f})$ denotes the number of nodes of fragment f, then $\sigma_{\mathsf{size}\leq 3}(\mathtt{F})$ will map only those fragments of F into *true*, whose $\mathsf{size}(\mathtt{f})$ does not exceed the value *3*. Clearly, this filter has anti-monotonic property because if $\mathsf{size}(\mathtt{f}) \leq 3$ is mapped into $false$, then all those fragments having $\mathsf{size}(\mathtt{f})$ value larger than *3* (super-fragments of f) will also be mapped into $false$. See Figure 6.

3.3.2 Height and Width of a Fragment

Logically interrelated components of a document are typically close to each other. Therefore, both the vertical and horizontal distance between nodes of a tree representing such a document are good measures of inter-relationship between nodes. It is fair to say that users are likely to judge fragments, in which horizontal or vertical distance between the nodes containing the query keywords exceed a certain threshold, be irrelevant. Supposing, $\mathsf{height}(\mathtt{f})$ denotes the vertical distance between the root and the farthest leaf node of the fragment f, then $\sigma_{\mathsf{height}\leq 2}(\mathtt{F})$ will map only those fragments of F into *true* whose $\mathsf{height}(\mathtt{f})$ does not exceed the value *2* (See Figure 6). Similar filter can be considered for eliminating fragments according to maximal horizontal distance between extreme nodes (the leftmost and the rightmost) of a fragment. Obviously, such filters also possess anti-monotonic property.

Of course, we can go on considering other filters having

anti-monotonic property and the list may be endless. However, our intention is not to create such a list but to give readers an insight into the existence of several filters that are practically useful and at the same time possess the anti-monotonic property.

3.4 Other Filters

One obvious question would be whether or not all filters have anti-monotonic property. A simple example of filters not having this property is a filter for all fragments consisting of nodes, whose number is greater than a certain value. We introduce another filter which looks more practically useful, which however, does not possess this property. We call this filter an 'equal depth filter'; it selects fragments in which each node having keyword k_1 is at the same vertical distance as the node having keyword k_2 from the root. In Figure 7, although the fragment f' does not satisfy the predicate defined by this filter, fragment f, which is a super-fragment of f' satisfies it. Therefore, it is clear that not all filters are anti-monotonic filters.

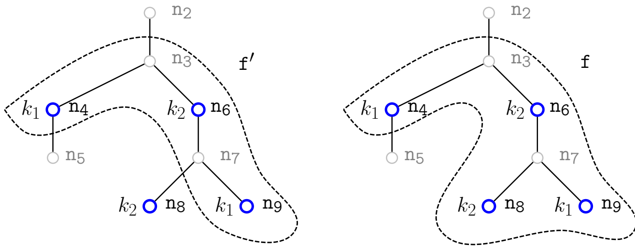

Figure 7: Filter not having Anti-monotonic Property

4. EXAMPLE OF QUERY EVALUATION

Having provided the theoretical foundation, we now illustrate how the operations we described in the previous sections can be applied to our running example query {XQuery, optimization} against the document shown in Figure 1. Remember our objectives are twofold: 1) to generate the target fragment (refer to Figure 8(b)), which, to the best of our knowledge, none of the existing techniques would have produced, and 2) to exclude irrelevant-looking fragments such as the one shown in Figure 8(c) at the earliest possible stage of query processing for saving the cost.

First, let $\mathcal{D}$, the tree shown in Figure 8(a), represents our example XML document of Figure 1 and $Q_{P_a}\{k_1, k_2\}$ denotes the example query, where $k_1 = \texttt{XQuery}$, $k_2 = \texttt{optimization}$ and P_a is an anti-monotonic filter. For this particular example, we choose this filter to be $\mathsf{size} \leq 3$.

Now, according to the explanation given in Section 2.3, this query can be evaluated as:

$$Q_{P_a}\{k_1, k_2\} \quad = \quad \sigma_{\mathsf{size} \leq 3}(F_1 \bowtie^* F_2)$$

where $F_1 = \sigma_{\mathsf{keyword}=XQuery}(F)$, $F_2 = \sigma_{\mathsf{keyword}=Optimization}(F)$ and $F = \texttt{nodes}(\mathcal{D})$. We have, $F_1 = \{\langle n_{17}\rangle, \langle n_{18}\rangle\}$ and $F_2 = \{\langle n_{16}\rangle, \langle n_{17}\rangle, \langle n_{81}\rangle\}$. For clarity, we write $\langle n_{16}\rangle$ as f_{16}, $\langle n_{17}\rangle$ as f_{17} and so on. We show how each evaluation strategy given below produces the same final result. We shall also explain why some strategies have more opportunities for optimization than the others.

4.1 Brute-Force Evaluation

This strategy does not consider any algebraic manipulation of the operations, and for this reason is the most inefficient way of producing a query result. However, for experimental evaluation of the model in future, it will provide the basis for performance comparison with other available alternative strategies.

Here, the answer set is computed, first, by directly applying the *powerset fragment join* operation between the two fragment sets F_1 and F_2 to generate a set of all computable fragments, and then, by applying the *selection* operation on this set for filtering irrelevant-looking fragments. Remember that *powerset fragment join* between F_1 and F_2 is equivalent to the *fragment join* on pairwise union of all non-empty subsets of F_1 and F_2. Following this definition, our example produces 11 unique pairwise unions (candidate fragment sets) on which we need to perform *fragment join* operation. These candidate fragments sets and the fragments produced by them after the *fragment join* operation are listed in Table 1 (column two and three respectively).

Note that some of the fragments, for example No.1 and No.8 in Table 1, are generated more than once. Among these 11 fragments, only the top seven (No.1-7 in Table 1) are unique. The last four at the bottom of Table 1 (No.8-11) are duplicates and will be removed from the set before performing the filter operation. Since our filter is $\mathsf{size} \leq 3$, only the first four fragments (No.1-4) will remain in the final answer set. Among these four, the first fragment represented by $\langle n_{16}, n_{17}, n_{18}\rangle$ is the fragment of interest, which we have successfully generated and consequently, met our first objective. The remaining three fragments (No.2-4) are *overlapping* answers which may or may not be of interest to users. See Section 5 for our views on overlapping answers.

4.2 Application of Set Reduction Technique

According to the definition of *fragment set reduce* (see Definition 10), $\ominus(F_2) = \{f_{17}, f_{81}\}$ while F_1 is already a reduced set as its cardinality is *2*. Hence, according to Theorem 1, F_1^+ and F_2^+, the corresponding *fixed points* of F_1 and F_2, can be computed by $F_1 \bowtie F_1$ and $F_2 \bowtie F_2$ respectively.

Note $F_1^+ = \{f_{17}, f_{18}, f_{17} \bowtie f_{18}\}$. Similarly, $F_2^+ = \{f_{16}, f_{17}, f_{81}, f_{16} \bowtie f_{17}, f_{16} \bowtie f_{81}, f_{17} \bowtie f_{81}\}$. Note also that the operation $F_1^+ \bowtie F_2^+$ produces the same 11 unique candidate fragment sets as listed in Table 1 (second column). It clearly illustrates that the expressions $F_1 \bowtie^* F_2$ and $F_1^+ \bowtie F_2^+$ are indeed equivalent.

What is not yet clear, however, is whether or not there will be a significant performance gain by applying this strategy instead of the one explained above. One important thing to note here is, unlike brute-force evaluation strategy, application of set reduction technique does offer an opportunity for optimization, because it avoids exhaustive means of computing all the candidate fragment sets that are to be joined. We believe by developing efficient algorithms that would require minimal overhead to compute *fragment set reduce*, we can expect a large performance gain — not in all cases — but in cases when fragment sets are estimated to be reduced by a factor over a certain numerical value. See Section 5 for more on estimating this value.

4.3 Introduction of an Anti-monotonic Filter

The idea here is to perform all the *selections* with anti-monotonic filters as early as possible. Consider $f_{16} \bowtie f_{81}$

Table 1: Input Fragment Sets and their Corresponding Output Fragments

No.	Fragment set to be joined	Fragment generated after join	Irrelevant (to be filtered)	Duplicate (to be removed)
1	$f_{17} \bowtie f_{18}$	$\langle n_{16}, n_{17}, n_{18} \rangle$		
2	$f_{16} \bowtie f_{17}$	$\langle n_{16}, n_{17} \rangle$		
3	$f_{16} \bowtie f_{18}$	$\langle n_{16}, n_{18} \rangle$		
4	f_{17}	$\langle n_{17} \rangle$		
5	$f_{17} \bowtie f_{81}$	$\langle n_0, n_1, n_{14}, n_{16}, n_{17}, n_{79}, n_{80}, n_{81} \rangle$	•	
6	$f_{18} \bowtie f_{81}$	$\langle n_0, n_1, n_{14}, n_{16}, n_{18}, n_{79}, n_{80}, n_{81} \rangle$	•	
7	$f_{17} \bowtie f_{18} \bowtie f_{81}$	$\langle n_0, n_1, n_{14}, n_{16}, n_{17}, n_{18}, n_{79}, n_{80}, n_{81} \rangle$	•	
8	$f_{16} \bowtie f_{17} \bowtie f_{18}$	$\langle n_{16}, n_{17}, n_{18} \rangle$		•
9	$f_{16} \bowtie f_{17} \bowtie f_{81}$	$\langle n_0, n_1, n_{14}, n_{16}, n_{17}, n_{79}, n_{80}, n_{81} \rangle$	•	•
10	$f_{16} \bowtie f_{18} \bowtie f_{81}$	$\langle n_0, n_1, n_{14}, n_{16}, n_{18}, n_{79}, n_{80}, n_{81} \rangle$	•	•
11	$f_{16} \bowtie f_{17} \bowtie f_{18} \bowtie f_{81}$	$\langle n_0, n_1, n_{14}, n_{16}, n_{17}, n_{18}, n_{79}, n_{80}, n_{81} \rangle$	•	•

which produces the fragment $\langle n_0, n_1, n_{14}, n_{16}, n_{79}, n_{80}, n_{81} \rangle$. Since this fragment does not satisfy the selection predicate of our filter size $\leq$ *3*, this and any other join involving $f_{16} \bowtie f_{81}$ (for example No.10 and No.11 in Table 1) can be ignored for further processing. Consequently, we can avoid several unnecessary join computations which would eventually produce irrelevant fragments. This clearly represents a great benefit in terms of amount of computation to be performed. Note that those fragment sets, which participate in producing relevant fragments in the final answer set, are never filtered out by anti-monotonic filters.

Particularly in a large XML tree, in which a significant number of potentially irrelevant fragments to a query may exist, this strategy of pushing down *selection* operations ahead of *join* operations will play a crucial role for gaining efficiency.

From the analysis given above, it should be clear that, 1) use of brute force strategy will make little sense in practical applications, 2) set reduction technique should be used if we can estimate that the fragment sets will be reduced by a factor over a certain value, and 3) more importantly, *selection* operations should be performed ahead of any *join* operations if the selection predicates possess an anti-monotonic property. These strategies should provide a strong foundation to achieve our second objective mentioned above.

5. DISCUSSIONS

In this section, we present several issues which we did not explore thoroughly in this paper. Without going into technical details, we shall keep our discussion simple and informal. First, we discuss the requirement of a cost model for estimating the cost of the operations defined in our query mechanism. Choice of an appropriate cost model is often implementation-dependent, yet, to prove the viability of our query model, simply presenting the techniques of logical query optimization may be inadequate. A formal cost model would also enable us to provide more authentic answers to issues such as the one raised in Section 3.1.4.

In [13], we investigated the feasibility of implementing our proposed model in a conventional relational database. With the assumption that we shall continue the relational database as our implementation platform, we plan to develop a cost model that also takes into account of these particular implementation-specific details, which will provide a basis for detecting various cost-based optimization strategies. Among many other things that a cost model has to consider, cost analysis of *fragment set reduce* operation, in particular, requires special attention for enabling judicious application of the set reduction technique (Section 4.2) to compute *fixed point* of a fragment set. We believe the cost of *fragment set reduce* operation on a fragment set F will depend not only on the cardinality of F, but also on the size of each fragment that each possible subset of F is going to produce. It will also depend upon the way we map underlying data into relational tables, the type of indices to be used, and more importantly, upon whether or not *fragment set reduce* will be able to take advantage of these indices.

Once we develop a suitable cost model, the next important issue would be to investigate the primary challenges that a query optimizer might face. Here, we give an insight into one such challenge. Before proceeding further into this discussion, let us informally define a term "reduction factor" that quantifies the amount of reduction obtained after *fragment set reduce* operation on a fragment set F. We denote this factor as RF and to keep this issue simple, let us suppose that RF can be computed as $RF = (a-b)/a$ where $a = \mid F \mid$ and $b = \mid \ominus(F) \mid$. Note that $0 >= RF < 1$, and $RF = 0$ means that no reduction has been obtained at all, while a higher value of RF means a significant reduction in the number of fragments in the fragment set. Through several experiments, suppose we can come up with a value v of RF such that, any value that is less than v implies there will be no benefit in applying *fragment set reduce* operation. The query optimizer can then estimate RF and compare it with v to decide whether or not to perform *fragment set reduce* operation for computing the *fixed point* of a fragment set. The challenge then for the optimizer would be to estimate RF accurately.

Before ending the discussion on this issue, we emphasize that further analysis is required both for devising a concrete cost model, and for designing a reliable query optimizer as well.

The second issue we are going to discuss is how to treat overlapping answers that we mentioned in Section 4.1. Note that overlapping answers here are similar to the ones that the IR-community has been debating in recent years[10][3]. There are contrasting views on whether or not some overlap in the results should be tolerated. An element-based XML retrieval system typically returns a set of ranked XML *ele-*

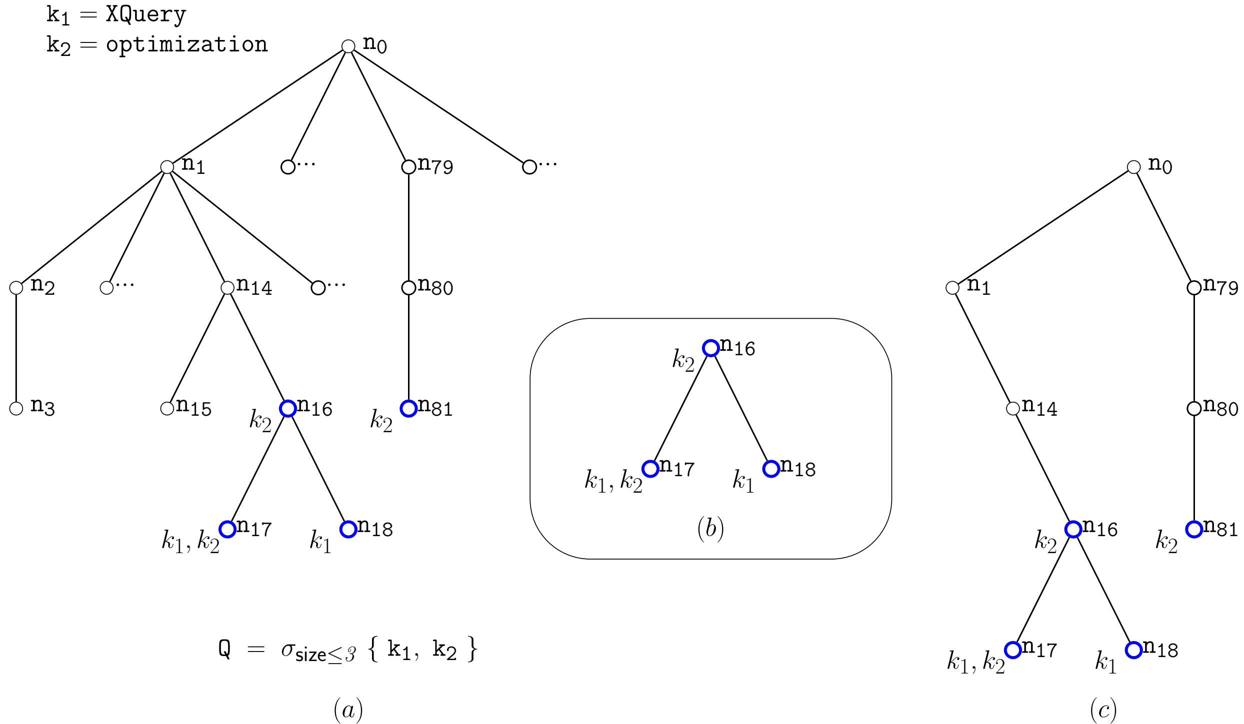

Figure 8: (a) An XML Tree and an Example Query, (b) Fragment of Interest, and (c) Potentially an Irrelevant Fragment

ments as a search result. In such a system, it may be wise to penalize overlapping results, otherwise the system might produce a result in which the top ranks are dominated by many structurally related elements[3]. In our case, overlapping answers are simply the sub-fragments of target fragments. We believe it is only a question of how they should be presented to the users. Either they can be completely hidden, or, together with target fragments, they can be presented in a visually pleasing way to show their structural relationships.

6. RELATED WORK

In contrast to complex syntax of structure-based query languages such as XQuery[19], keyword search offers a simple query interface to general users. In addition, it relieves users from being aware of the underlying structure of a document to formulate queries. Several studies on keyword search over XML documents have been done in the recent past. They can be classified according to 1) whether primary target documents are data-centric or document-centric, 2) the way query processing is carried out, and 3) whether or not any preprocessing of documents is carried out in order to build indices so that those indices can assist in efficient query processing.

Data-centric vs Document-centric Data: Several studies described in the literature[4][5][12][15][20] assume their target data to be highly schematic XML documents. Consequently, as explained in the Introduction, the query semantics proposed in these studies are not sufficient to effectively retrieve meaningful portions of less schematic document-centric XML documents even for simple queries.

Although studies described in [1][6][7][8][16] are focussed on keyword search over document-centric XML documents, none of them except [8] deals with the issue of *retrieval unit*. [16] provides extended SQL-like language for supporting both structure-based and IR-like keyword queries. XRank[7], largely influenced by Web-like search, proposes various techniques for ranking a large number of query results. [1][6] on the other hand, proposes methods for integrating keyword queries into conventional structural queries.

Database-style vs IR-style Query Processing: The idea of providing an interface as simple as “keyword search” for querying documents was originated by the IR community in the first place. Therefore, it is natural that several studies described in [7][5][1][6][17] adopt IR-style query processing. That is, instead of providing a strict query semantics, they provide several ranking techniques to deal with often overwhelming answers to a query. In contrast, we have limited our work within a framework which is purely based on database-style query processing, although ranking techniques described in those studies can be easily incorporated into our work. We provide a filtering mechanism, instead of ranking techniques, to restrict size of the query result. Our main interest lies in set-based operations and query optimization based on algebraic manipulation.

Preprocessing of Data: Some existing work [8][5][9] proposes preprocessing of the data for performance enhancement. In [8], it is claimed that several XML fragments, which may never be answer unit to a query, can be disregarded while building the index. This will help reduce the size of the index and thus enhance the performance of the system. Statistical measures have been taken into consid-

eration for identifying such irrelevant fragments. Our work differs from them in that no preprocessing of data is carried out and all answer fragments of interest are computed dynamically.

Direct comparison between the processing efficiency of existing approaches and the approach presented in this paper is difficult since existing methods are ineffective in achieving our goal in the first place. Our approach may fall short of efficiency under certain circumstances, as there is a natural trade-off between effectiveness and efficiency. However, we believe this trade-off can be compensated, at least partly, as we showed that there are several practically useful filters having anti-monotonic property which can be applied for better performance results. Our work can therefore serve as a complement to several other studies carried out for efficient keyword search over XML documents.

The problem related to retrieval unit for keyword-based queries has been studied by many researchers in various contexts. [2] deals with this problem in the context of relational database management systems while [11] does that in the context of logical web documents consisting of multiple pages. In either of these papers, the underlying data is modelled as a graph and heuristic methods have been proposed to find out the most relevant answers from the graph. Both suffer from the problem of computational complexity. We have stated the problem of computing retrieval unit in the context of document-centric XML documents, and provided a query model that can compute potentially relevant answers effectively and efficiently under certain circumstances.

7. CONCLUSIONS AND FUTURE WORK

A theoretical framework for an effective and efficient keyword search over document-centric XML documents was described. We showed that even for simple queries, existing approaches are not always effective in retrieving fragments of interest. Opportunity for optimization is sought by any query mechanism to reduce the query-processing cost. In this paper, we presented several optimization techniques that guarantees better efficiency for keyword search over tree-structured documents. We provided several analysis and theoretical proof which would allow a query processor to manipulate operations algebraically. Moreover, we defined a particular class of filters having anti-monotonic property. We theoretically showed that these filters, when used as selection predicates, are capable of reducing significant processing cost. The model can be easily implemented on top of an existing relational database and hence, can accommodate a very large collection of XML documents[13].

Experimental evaluation involving real data sets, algorithms to implement all the operations and their precise complexity analyses, answer presentation techniques for *overlapping* answers considered natural in document-centric XML documents, and other optimization issues at implementation level to complement our algebraic optimization are some of the issues for our future work.

Acknowledgements

The author is grateful to Professor Katsumi Tanaka of Kyoto University for his encouragement in initiating this work and for suggestions he provided through numerous discussions. The author is also thankful to the anonymous referees for their invaluable comments and suggestions.

8. REFERENCES

[1] Shurug Al-Khalifa, Cong Yu, and H. V. Jagadish. Querying structured text in an XML database. In *SIGMOD 2003*, pages 4–15, 2003.

[2] G. Bhalotia, C. Nakhe, A. Hulgeri, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, pages 431–440, 2002.

[3] Charles L. A. Clarke. Controlling overlap in content-oriented XML retrieval. In *SIGIR*, pages 314–321, 2005.

[4] S. Cohen, Y. Kanza, and B. Kimelfeld. Interconnection semantics for keyword search in XML. In *Proc. of CIKM*, pages 389–396, 2005.

[5] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A semantic search engine for XML. In *Proc. of 29th VLDB*, pages 45–56, 2003.

[6] D. Florescu, D. Kossman, and I. Manolescu. Integrating keyword search into XML query processing. In *International World Wide Web Conference*, pages 119–135, 2000.

[7] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRank: ranked keyword search over XML documents. In *SIGMOD*, pages 16–27. ACM, June 2003.

[8] K. Hatano, H. Kinutani, T. Amagasa, Y. Mori, M. Yoshikawa, and S. Uemura. Analyzing the properties of XML fragments decomposed from the INEX document collection. In *INEX*, pages 168–182, 2004.

[9] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *ICDE*, pages 367–378. IEEE, 2003.

[10] G. Kazai, M. Lalmas, and A. P. de Vries. The overlap problem in content-oriented XML retrieval evaluation. In *SIGIR*, pages 72–79, 2004.

[11] W.S. Li, K. S. Candan, Q. Vu, and D. Agrawal. Retrieving and organizing web pages by 'Information Unit'. In *Tenth International WWW Conference, Hong Kong, China*, pages 230–244, 2001.

[12] Y. Li, C. Yu, and H. V. Jagadish. Schema-free XQuery. In *Proc. of 30th VLDB*, pages 72–83, 2004.

[13] S. Pradhan. A framework for the relational implementation of tree algebra to retrieve structured document fragments. In *5th Int'l Conf. on Web Information Systems Engineering*, pages 206–217. Springer-Verlag, Nov 2004.

[14] S. Pradhan and K. Tanaka. Retrieval of relevant portions of structured documents. In *Proc. 15th Int'l Conf. of Database and Expert Systems Applications*, pages 328–338. Springer-Verlag, Aug-Sep 2004.

[15] A. Schmidt, M. Kersten, and M. Windhouwer. Querying XML documents made easy: Nearest concept queries. In *ICDE*, pages 321–329, 2001.

[16] A. Theobald and G. Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *EDBT 2002: 8th International Conference on Extending Database Technology*, pages 477–495. Springer-Verlag, 2002.

[17] Martin Theobald, Ralf Schenkel, and Gerhard Weikum. An efficient and versatile query engine for TopX search. In *VLDB*, pages 625–636, 2005.

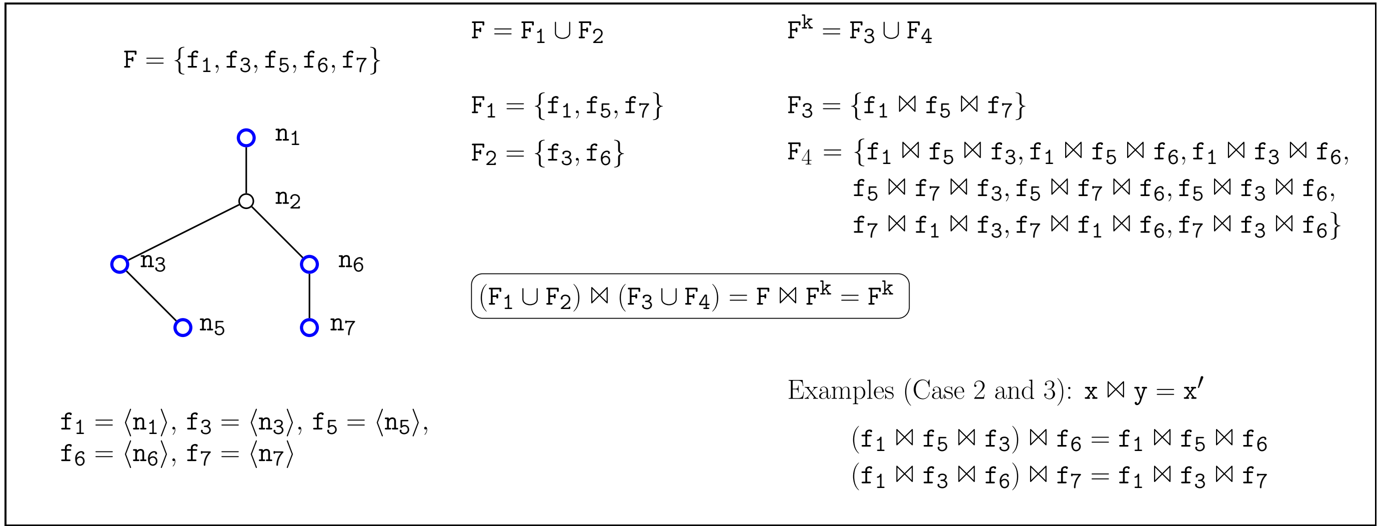

Figure 9: Illustrative Proof for Theorem 1

[18] J. D. Ullman. *Principles of Database and Knowledge-Base Systems Vol. II.* Computer Science Press, 1989.

[19] W3C. XQuery 1.0: An XML query language. http://www.w3.org/TR/xquery/.

[20] Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD*, pages 527–538. ACM, June 2005.

Appendix

Proof Theorem 1. For $\mid F \mid \; <= 2$, the proof is trivial, since for any fragment set to be reduced, the set should contain at least three elements. In order to prove $\bowtie_n (F) = \bowtie_k (F)$, we only need to prove $\bowtie_k (F) = \bowtie_{k+1} (F) = \bowtie_k (F) \bowtie F$. We shall prove this by showing $\bowtie_k (F) \subseteq \bowtie_k (F) \bowtie F$ and $\bowtie_k (F) \bowtie F \subseteq \bowtie_k (F)$.

Here, $\bowtie_k (F) \subseteq \bowtie_k (F) \bowtie F$ is obvious because of monotonic property of *pairwise fragment join* operation. However, we need further analysis of the operation in order to prove $\bowtie_k (F) \bowtie F \subseteq \bowtie_k (F)$. Let F^k denotes the fragments produced in the kth iteration, that is, $F^k = \bowtie_k (F) - \bowtie_{k-1} (F)$. We now only need to show $F^k \bowtie F \subseteq \bowtie_k (F)$.

Let F_1 denotes the reduced set of F and $F_2 = F - F_1$. We know, F^k may contain at most $\frac{n!}{k!(n-k)!}$ (combinations of n-fragments taken k-fragments at a time) number of element fragments. Let F_3 denotes $\{f^1 \bowtie f^2 \bowtie \ldots \bowtie f^k\}$ where f^i is the ith element of F_1. Obviously, $\{f^1 \bowtie f^2 \bowtie \ldots \bowtie f^k\} \in F^k$. Supposing $F_4 = F^k - F_3$ then we must show that $(F \bowtie F_3) \cup (F_1 \bowtie F_4) \cup (F_2 \bowtie F_4) \subseteq F^k$ because
$F \bowtie F^k = (F_1 \cup F_2) \bowtie (F_3 \cup F_4) =$
$(F_1 \bowtie F_3) \cup (F_1 \bowtie F_4) \cup (F_2 \bowtie F_3) \cup (F_2 \bowtie F_4) =$
$(F \bowtie F_3) \cup (F_1 \bowtie F_4) \cup (F_2 \bowtie F_4)$. Now, we analyse all three cases.

Case 1 $(F \bowtie F_3)$. *Obviously, all the fragments in* F *are the sub-fragments of* F_3 *since* F_3 *is a singleton consisting of the largest fragment that could be generated from the elements in* F. *Therefore,* $F \bowtie F_3 \subseteq F^k \subseteq \bowtie_k (F)$.

Case 2 $(F_2 \bowtie F_4)$. *Let* $x \in F_4$, $y \in F_2$ *and* $y', y'' \in F$ $(y \neq y' \neq y'')$ *such that* $y \subseteq y' \bowtie y''$. *Now, if* $y \subseteq x$, *then* $x \bowtie y = x$. *If* $y \not\subseteq x$, *there must exist* $x' \in F^k$ *such that* $y' \subseteq x'$ *and* $y'' \subseteq x'$ *because* F^k *consists of all k-way combinations of n fragments of* F *and for each fragment* f *in* F_4 *there exists* $f' \in F_2$ *such that* $f' \subseteq f$. *It implies,* $x' = x \bowtie y$ *(refer to Figure 9). Therefore,* $x \bowtie y \in F^k$ *which follows* $F_2 \bowtie F_4 \subseteq F^k \subseteq \bowtie_k (F)$.

Case 3 $(F_1 \bowtie F_4)$. *Let* $x \in F_4$, $y \in (F_1)$ *and* $y', y'' \in F$ $(y \neq y' \neq y'')$ *such that* $y \subseteq y' \bowtie y''$. *Now, if* $y \subseteq x$, *then* $x \bowtie y = x$. *If* $y \not\subseteq x$, *there must exist* $x' \in F^k$ *such that either 1.* $(y \subseteq x'$ *and* $y' \subseteq x')$ *or 2.* $(y \subseteq x'$ *and* $y'' \subseteq x')$ *because* F^k *consists of all k-way combinations of n fragments of* F *and for each fragment* f *in* F_4 *there exists* $f' \in F_2$ *such that* $f' \subseteq f$. *It implies,* $x' = x \bowtie y$ *(refer to Figure 9). Therefore,* $x \bowtie y \in F^k$ *which follows* $F_1 \bowtie F_4 \subseteq F^k \subseteq \bowtie_k (F)$.

This proves the theorem. ☐

Proof Theorem 3. We prove this theorem by showing $\sigma_{P_a}(\sigma_{P_a}(F_1) \bowtie \sigma_{P_a}(F_2)) \subseteq \sigma_{P_a}(F_1 \bowtie F_2)$ and $\sigma_{P_a}(F_1 \bowtie F_2) \subseteq \sigma_{P_a}(\sigma_{P_a}(F_1) \bowtie \sigma_{P_a}(F_2))$. In order to do that, first we define the following lemma.

Lemma 1. *A fragment* f *is the sub-fragment of the resulting fragment produced by the* fragment join *operation between* f *and any arbitrary fragment* f', *that is,* $f \subseteq f \bowtie f'$.

Proof. Obvious from the definition of *fragment join.* ☐

We have $\sigma_{P_a}(F_1) \subseteq F_1$ and $\sigma_{P_a}(F_2) \subseteq F_2$ from the definition of *selection.* It follows $\sigma_{P_a}(F_1) \bowtie \sigma_{P_a}(F_2) \subseteq F_1 \bowtie F_2$. Therefore, $\sigma_{P_a}(\sigma_{P_a}(F_1) \bowtie \sigma_{P_a}(F_2)) \subseteq \sigma_{P_a}(F_1 \bowtie F_2)$.

Now, let $f \in \sigma_{P_a}(F_1 \bowtie F_2)$. Since $\sigma_{P_a}(f) = true$, there must be $f_1 \in F_1$ and $f_2 \in F_2$ such that $f_1 \bowtie f_2 = f$. Since $f_1 \subseteq f$ according to Lemma 1, and P_a is an anti-monotonic filter, it follows $\sigma_{P_a}(f_1) = true$. Therefore, $f_1 \in \sigma_{P_a}(F_1)$. Similarly, $f_2 \in \sigma_{P_a}(F_2)$. As $\sigma_{P_a}(f) = true$ and $f_1 \bowtie f_2 = f$, it follows $f_1 \bowtie f_2 \in \sigma_{P_a}(\sigma_{P_a}(F_1) \bowtie \sigma_{P_a}(F_2))$. Hence, $\sigma_{P_a}(F_1 \bowtie F_2) \subseteq \sigma_{P_a}(\sigma_{P_a}(F_1) \bowtie \sigma_{P_a}(F_2))$. ☐