Quantifiers in XQuery

Norman May, Sven Helmer, and Guido Moerkotte
Universität Mannheim
Mannheim, Germany
norman, helmer, moer@pi3.informatik.uni-mannheim.de

Abstract

We present algebraic equivalences that allow to unnest nested algebraic expressions containing quantifiers for order-preserving algebraic operators. We illustrate how these equivalences can be applied successfully to unnest nested queries formulated in XQuery. Measurements illustrate the performance gains possible by unnesting.

1. Introduction

Quantification is a core feature of XQuery, the query language for XML data proposed by the W3C.[1] The keywords `some` and `every` are used to express existential quantification and universal quantification.

Query optimization for queries containing quantification has been investigated in the relational and object-oriented context — see [2] for related work. Initially, unnesting techniques were performed on the source level. However, now researchers prefer to describe unnesting equivalences on the algebraic level, because such techniques remain valid independent of the query language as long as it can be translated into the algebra. In addition, correctness proofs can be provided for the equivalences. Thus, we define our unnesting techniques on the algebraic level also.

However, if the result's order is relevant, the unnesting techniques from the object-oriented and relational context cannot be applied. We show how to unnest XML queries containing quantifiers.

The paper is organized as follows. Section 2 briefly motivates and defines our algebra. For every unnesting equivalence presented in Section 3, we show how the equivalences are applied and provide performance figures for each evaluation plan. Section 4 concludes the paper.

2. Notation and Algebra

Our algebra works on sequences of sets of variable bindings, i.e., sequences of unordered tuples where every attribute corresponds to a variable. We allow nested tuples, i.e. the value of an attribute may be a sequence of tuples. Single tuples are constructed using the standard $[\cdot]$ brackets. Concatenation of tuples and functions is denoted by $\circ$. The set of attributes defined for an expression e is defined as $\mathcal{A}(e)$. The set of free variables of an expression e is defined as $\mathcal{F}(e)$.

Projection of a tuple on a set of attributes A is denoted by $|_A$. For an expression e_1 possibly containing free variables, and a tuple e_2, we denote by $e_1(e_2)$ the result of evaluating e_1 where bindings of free variables are taken from variable bindings provided by e_2. Of course, this requires $\mathcal{A}(e_2) \subseteq \mathcal{F}(e_1)$. For a set of attributes we define the tuple constructor $\perp_A$ such that it returns a tuple with attributes in A initialized to NULL.

For sequences e, we use $\alpha(e)$ to denote the first element of a sequence. We equate elements with single element sequences. The function τ retrieves the tail of a sequence and $\oplus$ concatenates two sequences. We denote the empty sequence by ϵ.

Our algebra extends the SAL-Algebra [1] developed by Beeri and Tzaban. SAL is the order-preserving counterpart of the algebra used in [3] extended to handle semistructured data. We give definitions for all relevant algebraic operators – including our new result constructing operator Ξ – to make the paper self-contained.

In figure 1 we define the algebraic operators recursively on their input sequences. For unary operators, if the input sequence is empty, the output sequence is also empty. For binary operators, the output sequence is empty whenever the left operand represents an empty sequence.

In order to avoid special cases during the translation of XQuery into the algebra, we use the special algebraic operator ($\Box$) that returns a singleton sequence consisting of the empty tuple, i.e. a tuple with no attributes.

We also define a duplicate eliminating projection Π^D_A. Besides the projection, it has similar semantics as the `distinct-values` function of XQuery: it does not preserve order. However, we require it to be deterministic and idempotent. When we want to eliminate the set of attributes

[1] http://www.w3.org/XML/Query

$$\sigma_p(e) := \begin{cases} \alpha(e) \oplus \sigma_p(\tau(e)) & \text{if } p(\alpha(e)) \\ \sigma_p(\tau(e)) & \text{else} \end{cases}$$

$$\Pi_A(e) := \alpha(e)|_A \oplus \Pi_A(\tau(e))$$

$$e[a] := [a : \alpha(e)] \oplus \tau(e)[a]$$

$$\chi_{a:e_2}(e_1) := \alpha(e_1) \circ [a : e_2(\alpha(e_1))] \oplus \chi_{a:e_2}(\tau(e_1))$$

$$\mu_g(e) := (\alpha(e)|_{\overline{\{g\}}} \times \alpha(e).g) \oplus \mu_g(\tau(e))$$

$$\Upsilon_{a:e_2}(e_1) := \mu_g(\chi_{g:e_2[a]}(e_1))$$

$$\Gamma_{g;\theta A;f}(e) := \Pi_{A:A'}(\Pi^D_{A':A}(\Pi_A(e))\Gamma_{g;A'\theta A;f}e)$$

$$e_1 \times e_2 := (\alpha(e_1)\overline{\times}e_2) \oplus (\tau(e_1) \times e_2) \text{ where}$$

$$e_1\overline{\times}e_2 := \begin{cases} \epsilon & \text{if } e_2 = \epsilon \\ (e_1 \circ \alpha(e_2)) \oplus (e_1\overline{\times}\tau(e_2)) & \text{else} \end{cases}$$

$$e_1 \ltimes_p e_2 := \begin{cases} \alpha(e_1) \oplus (\tau(e_1) \ltimes_p e_2) & \text{if } \exists x \in e_2 \;\; p(\alpha(e_1) \circ x) \\ \tau(e_1) \ltimes_p e_2 & \text{else} \end{cases}$$

$$e_1 \triangleright_p e_2 := \begin{cases} \alpha(e_1) \oplus (\tau(e_1) \triangleright_p e_2) & \text{if } \nexists x \in e_2 \;\; p(\alpha(e_1) \circ x) \\ (\tau(e_1) \triangleright_p e_2) & \text{else} \end{cases}$$

$$e_1\Gamma_{g;A_1\theta A_2;f}e_2 := \alpha(e_1) \circ [g : G(\alpha(e_1))] \oplus (\tau(e_1)\Gamma_{g;A_1\theta A_2;f}e_2)$$

Figure 1. Algebraic Operators

A, we denote this by $\Pi_{\overline{A}}$. We use $\Pi_{A':A}$ for renaming attributes. The attributes in A are renamed to those in A'. Attributes other than those in A remain untouched.

The grouping operators Γ produce a sequence-valued new attribute g containing "the group" where $G(x) := f(\sigma_{x|_{A_1}\theta A_2}(e_2))$ and $\theta \in \{=, \leq, \geq, <, >, \neq\}$. Unary grouping is defined by using a binary grouping operator.

The unnest operator μ retrieves the sequence of tuples of attribute g. If g is empty, it returns the tuple $\perp_{\mathcal{A}(e.g)}$. The unnest map operator Υ is mainly used for evaluating XPath expressions. Since this is a very complex issue [5, 6], we do not delve into optimizing XPath evaluation, but instead take an XPath expression occurring in a query as is and use it in the place of e_2. An optimized translation is well beyond the scope of the paper. But we would also like to point out that the Υ operator generates its output in document order if the translation of XPath expressions described in [6] is used.

Our new result construction operator Ξ combines a pair of Groupify-GroupApply operators [4]. It executes a semicolon separated list of commands and, as a side effect, constructs the query result. For simplicity, we assume that the result is constructed as a string on some output stream. If a complex expression e evaluates to a sequence of tuples containing a string-valued attribute a that is successively bound to author names from some bibliography document, $\Xi_{"<author>";a;"</author>"}(e)$ embeds every author name into an `author` element.

3. Example Applications

In this section we present example applications for the unnesting equivalences (Fig. 2, see [7] for proofs). We based the queries on those in the XQuery use case document and the DTDs therein.[2] We rewrote the queries by renaming variables and simplifying them slightly, thereby retaining the essence of the query. Due to space restrictions, we omit our normalization and translation steps and describe our unnesting techniques only.

[2] http://www.w3.org/TR/xquery-use-cases/, use case XMP

$$\sigma_{\exists x \in (\Pi_{x'}(\sigma_{A_1=A_2}(e_2)))p}(e_1) = e_1 \ltimes_{A_1=A_2 \wedge p'} e_2 \quad (1)$$

$$\sigma_{\forall x \in (\Pi_{x'}(\sigma_{A_1=A_2}(e_2)))p}(e_1) = e_1 \triangleright_{A_1=A_2 \wedge \neg p'} e_2 \quad (2)$$

$$\Pi^D(e_1) \ltimes_{A_1=A_2} (\sigma_p(e_2)) = \sigma_{c>0}(E) \quad (3)$$

$$\Pi^D(e_1) \triangleright_{A_1=A_2} (\sigma_p(e_2)) = \sigma_{c=0}(E) \quad (4)$$

Figure 2. Unnesting Equivalences

Although the equivalences together with the conditions in Fig. 2 are not easy to grasp on a first reading, *correct* unnesting crucially depends on a rigorous treatment. Too often, incorrect unnesting procedures have appeared.

Before commenting on the equivalences, let us give the conditions that ensure correctness. For equivalences 1 and 2, $x' \in \mathcal{A}(e_2)$ must hold. Further, p' results from p by replacing x by x'. In equivalences 3 and 4 we assume $E = \Pi_{A_1:A_2}(\Gamma_{c;=A_2;count\circ\sigma_p}(e_2))$. The equivalences hold if $A_i \subseteq \mathcal{A}(e_i)$, $\mathcal{F}(e_2) \cap \mathcal{A}(e_1) = \emptyset$, and $\Pi^D(e_1) = \Pi^D_{A_1:A_2}(\Pi_{A_2}(e_2))$.

Why are the equivalences useful? In case the **where** clause contains a quantifier, the translation process results in an expression matching the left-hand side of one of the first two equivalences. We gain efficiency by rewriting them to the right hand side because evaluation of expression e_2 and predicate p' can be done independent from expression e_1. Instead of a nested loop evaluation with multiple scans over the same data we need to read the data sources only once. The last two equivalences save another scan when the data sources of both expressions are the same. They are typically applied after unnesting.

3.1. Existential Quantification I

Existential quantification in XQuery can be expressed in a quantified expression in the **where** clause as shown in the first example below. Note that although order preservation within the quantifier is not needed, the following query retrieves `title` elements in document order.

```
let $d1 := doc("bib.xml")
```

```
for $t1 in $d1//book/title
where some $t2 in (
         let $d2 := doc("reviews.xml")
         for $t3 in $d2//entry/title
         return $t3 )
      satisfies $t1 = $t2
return
  <book-with-review>
     { $t1 }
  </book-with-review>
```

The translation algorithm maps the **for** clause to the Υ operator and the **let** clause to the χ operator. The subscript of these operators is defined by the binding expression of both clauses. (Frequently, these binding expressions contain XPath expressions. However, since we concentrate on unnesting techniques in XQuery in this paper, we rely on efficient translation and evaluation techniques [5, 6]. These techniques can be used orthogonally to the methods presented in this paper.) The **where** clause is translated into a σ operator and the Ξ operator constructs the query result as defined in the **return** clause. We greatly simplify the translation of the **return** clause and refer to [4] for a more advanced treatment of result construction in XQuery.

In the first example we can move the correlation predicate into the range expression and translate the normalized query into $\Xi_{s1;t1;s2}(\sigma_{\exists t2 \in e_2}\,\text{true}(e_1))$

where

$$\begin{aligned} e_1 &:= \Upsilon_{t1:d1//book/title}(\chi_{d1:doc1}(\Box)) \\ e_2 &:= \Pi_{t3}(\sigma_{t1=t3}(e_3)) \\ e_3 &:= \Upsilon_{t3:d2//entry/title}(\chi_{d2:doc2}(\Box)) \end{aligned}$$

and

```
doc1 = doc("bib.xml")
doc2 = doc("reviews.xml")
s1   = "<book-with-review>"
s2   = "</book-with-review>"
```

We use Eqv. 1 to get

$$\Xi_{s1;t1;s2}(e_1 \ltimes_{t1=t3} e_3).$$

The performance of these two evaluation plans is compared for varying numbers of books and reviews as indicated in the following table. All experiments were carried out on a simple PC with a 2.4 Ghz Pentium. We observe that the unnested evaluation plan scales far better with larger input values. We note that the nested query is evaluated each time the outer query reads a tuple while the unnested plan scans each input document just once.

	Evaluation Time (#books, #reviews)		
Plan	100	1000	10000
nested	0.10 s	1.83 s	175.80 s
semijoin	0.08 s	0.09 s	0.20 s

3.2 Existential Quantification II

Existential quantification might be expressed in different ways. Instead of using a quantified expression, it is also possible to use the function `empty` or check if counting evaluates to zero. The following example illustrates a third alternative using the function `exists`.

```
let $d1 := doc("bib.xml")
for $b1 in $d1//book,
    $a1 in $b1/author
where exists(
        let $b2 := $d1//book
        for $a2 in $b2/author
        where contains($a2, "Suciu")
          and $b1 = $b2
        return $b2)
return
  <book>
     { $a1 }
  </book>
```

The translation of the query yields:

$$\Xi_{s1;a1;s2}(\sigma_{\exists b3 \in e_3}\,\text{true}(e_1))$$

where

$$\begin{aligned} e_1 &:= \Upsilon_{a1:b1/author}(\Upsilon_{b1:d1//book}(\chi_{d1:doc1}(\Box))) \\ e_2 &:= \Upsilon_{a2:b2/author}\Upsilon_{b2:d1//book}(\chi_{d1:doc1}(\Box)) \\ e_3 &:= \Pi_{a2}(\sigma_{b1=b2 \wedge contains(a2,"Suciu")}(e_2)) \end{aligned}$$

and

```
d1 = doc("bib.xml")
s1 = "<book>"
s2 = "</book>"
```

The expression can be unnested using Eqv. 1. We check the prerequisites of Eqv. 3 and notice that the required duplicate elimination on the `book` elements of e_1 is not necessary because `//book` returns a duplicate free sequence of books by definition. So we can apply Eqv. 3. Both expressions (for Eqv. 1 and Eqv. 3) are shown below.

$$\Xi_{s1;a2;s2}(e_1 \ltimes_{b1=b2 \wedge contains(a2,"Suciu")}(e_2)$$
$$\Xi_{s1;a2;s2}(\sigma_{c>0}(\Gamma_{c;=b2;counto\sigma_{contains(a2,"Suciu")}}(e_2))$$

In the table below, we summarize the execution times for the three presented expressions. The tremendous effect of unnesting can also be seen in this case. In addition, we observe a performance gain in the third evaluation plan, which is caused by avoiding one scan of the input document.

	Evaluation Time (#books)		
Plan	100	1000	10000
nested	0.04 s	1.31 s	138.8 s
semijoin	0.03 s	0.05 s	0.30 s
grouping	0.02 s	0.02 s	0.02 s

3.3. Universal Quantification

Besides existential quantification, XQuery supports universal quantification. The following example returns the authors whose books were all published after 1993.

```
let $d1 :=  doc("bib.xml")
for $a1 in distinct-values($d1//author)
where every $y2 in (
        let $d3 := doc("bib.xml")
        for $b3 in $d3//book
        let $y3 := $b3/@year
        for $a3 in $b3/author
        where $a1 = $a3
        return $y3 )
      satisfies $y2 > 1993
return
  <new-author>
     { $a1 }
  <new-author>
```

The nested query plan is derived by application of the translation rules where the function `distinct-values` is mapped to the Π^D operator.

$$\Xi_{s1;a1;s2}(\sigma_{\forall y2 \in e2\ y2>1993}(e_1))$$

where

$$\begin{aligned} e_1 &= \Upsilon_{a1:\Pi^D(d1//author)}(\chi_{d1:doc}(\Box)) \\ e_2 &= \Pi_{y3}(\sigma_{a1=a3}(e_3)) \\ e_3 &= \Upsilon_{a3:b3/author}(\chi_{y3:b3/@year}(\Upsilon_{b3:d3//book}(\\ &\quad \chi_{d3:doc}(\Box)))) \end{aligned}$$

and

```
doc = doc("bib.xml")
s1  = "<new-author>"
s2  = "</new-author>"
```

Eqv. 2 is applicable and yields

$$\Xi_{s1;a1;s2}(e_1 \triangleright_{a1=a3 \wedge y3 \leq 1993}\ e_3)$$

Of course, we can push the second part of the join predicate into its second operand. This yields

$$\Xi_{s1;a1;s2}(e_1 \triangleright_{a1=a3}\ \sigma_{y3 \leq 1993}(e_3))$$

Since we know from the DTD that `author` elements occur only under `book` elements, $\Pi^D_{a1}(e_1) = \Pi^D_{a1:a_3}(\Pi_{a_3}(e_3))$ holds and, thus, we can apply Eqv. 4, which yields:

$$\Xi_{s1;a1;s2}(\sigma_{c=0}(\Gamma_{c;=aa;count \circ \sigma_{y3 \leq 1993}}(e_3)))$$

A comparison of the evaluation times of the discussed plans is given in the table below. Again unnesting is highly beneficial. But the transformation using grouping does not boost the evaluation time as in the previous example.

Plan	Evaluation Time (#books) 100	1000	10000
nested	0.12 s	4.86 s	507.85 s
anti-semijoin	0.07 s	0.08 s	0.24 s
grouping	0.07 s	0.08 s	0.23 s

4 Conclusion

In the core of the paper we presented equivalences that allow to unnest nested algebraic expressions. Some of these equivalences are counterparts of existing equivalences valid for algebras whose operators do not preserve order. For others, no counterpart has been published so far. We demonstrated each of the equivalences by means of an example. Further, we experimentally compared the performance of the nested algebraic expressions with the unnested algebraic expressions. In doing so, enormous performance improvements could be observed.

Acknowledgment. We thank Simone Seeger for her help in preparing the manuscript and C.-C. Kanne for his help in carrying out the experiments.

References

[1] C. Beeri and Y. Tzaban. SAL: An algebra for semistructured data and XML. In *ACM SIGMOD Workshop on the Web and Databases (WebDB)*, 1999.

[2] J. Claussen, A. Kemper, G. Moerkotte, and K. Peithner. Optimizing queries with universal quantification in object-oriented and object-relational databases. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 286–295, 1997.

[3] S. Cluet and G. Moerkotte. Nested queries in object bases. In *Proc. Int. Workshop on Database Programming Languages*, 1993.

[4] T. Fiebig and G. Moerkotte. Algebraic XML construction and its optimization in Natix. *World Wide Web Journal*, 4(3):167–187, 2002.

[5] G. Gottlob, C. Koch, and R. Pichler. Xpath query evaluation: Improving time and space efficiency. In *Proc. IEEE Conference on Data Engineering*, page to appear, 2003.

[6] S. Helmer, C.-C. Kanne, and G. Moerkotte. Optimized translation of xpath expressions into algebraic expressions parameterized by programs containing navigational primitives. In *Proc. Int. Conf. on Web Information Systems Engineering (WISE)*, 2002. 215-224.

[7] N. May, S. Helmer, and G. Moerkotte. Nested queries and quantifiers in an ordered context. Technical report, University of Mannheim, 2003.