On the role of composition in XQuery

Christoph Koch
Lehrstuhl für Informationssysteme
Universität des Saarlandes
Saarbrücken, Germany
koch@cs.uni-sb.de

ABSTRACT

Nonrecursive XQuery is known to be hard for nondeterministic exponential time. Thus it is commonly believed that any algorithm for evaluating XQuery has to require exponential amounts of working memory and doubly exponential time in the worst case. In this paper we present a property – the lack of a certain form of composition – that virtually all real-world XQueries have and that allows for query evaluation in singly exponential time and polynomial space. Still, we are able to show for an important special case – our nonrecursive XQuery fragment restricted to atomic value equality – that the composition-free language is just as expressive as the language with composition. Thus, under widely-held complexity-theoretic assumptions, the composition-free language is an exponentially less succinct version of the language with composition.

1. INTRODUCTION

XQuery is the principal data transformation query language for XML. Full XQuery is Turing-complete, but queries without recursion are guaranteed to terminate in straightforward functional implementations of the XQuery language. Recursion in XQuery is rarely used in practice (see also [15]); recursive XML transformations are usually implemented in XSLT. It was shown in [7] that nonrecursive XQuery is NEXPTIME-hard. As a consequence, it is commonly believed that any query evaluation algorithm for nonrecursive XQuery must consume doubly exponential time and exponential space for query evaluation in the worst case (cf. e.g. [6]). This is by an exponential factor worse than the complexity of relational algebra or calculus [11]. The paper [7] also introduced a clean fragment of nonrecursive XQuery called *Core XQuery* that was shown to be the expressive counterpart of nested relational algebra [5] (or similar languages such as complex value algebra without powerset [1] or monad algebra [12]) for XML.

In this paper we present a syntactic property – the lack of a certain form of composition – that virtually all real-world XQueries have and which renders *composition-free* Core XQuery just as hard as relational algebra.

By composition, informally, we refer to the use of data value construction (rather than selection using XPath) anywhere except for the construction part of an XQuery (that is, in a for-let-where-return (FLWR) construct, anywhere except for the return clause). For example, the query

```
<books_2004>
{ for $x in /bib/book where year=2004 return
     <book>
        {$x/title}
        <authors>
           { for $y in $x/author return
               <author> {$y/lastname} </author>
           }
        </authors>
     </book>
}
</books_2004>
```

is a composition-free query (so nesting queries, e.g., FLWR-statements in their return clauses, is not a problem) while

```
<books>
{ let $x := <a>{ for $w in /bib/book
                 return <b> {$w} </b> }</a>
  for $y in $x/b return $y/*
}
</books>
```

is not composition-free because it uses a let-expression that constructs a tree as an intermediate result. The equivalent query

```
<books>
{ for $y in (for $w in /bib/book
             return <b> {$w} </b>)
  return $y/*}
}
</books>
```

is still not composition-free because the "in"-expression of the outer for-loop contains a for-loop. However, there is an equivalent composition-free query,

```
<books>
   { for $w in /bib/book return $w }
</books>
```

The technical contributions of this paper are as follows.

- It is shown that composition-free Core XQuery can be evaluated in polynomial space and thus also in singly exponential time. In fact, composition-free nonrecursive XQuery is PSPACE-complete.

With composition [7]:

	with negation	without negation
deep equality	in EXPSPACE; TA$[2^{O(n)}, O(n)]$-hard	
equality on atomic values	TA$[2^{O(n)}, O(n)]$-complete	NEXPTIME-complete

Without composition:

	with negation	without negation
deep equality	PSPACE-complete	NP-complete
equality on atomic values	PSPACE-complete	NP-complete

Table 1: Summary of results on query/combined complexity of Core XQuery.

- We also show that composition-free nonrecursive XQuery without negation is NP-complete.

- Still, we are able to show for an important special case – equality is restricted to atomic values – that composition-free Core XQuery is just as expressive as Core XQuery with composition. Thus, under the usual complexity-theoretic assumptions, the composition-free language is an exponentially less succinct version of the language with composition.

An overview of the complexity results of this paper – together with the results from [7] is given in Table 1. Since the variables in composition-free XQuery range only over subtrees of the input tree, supporting deep value equality has no influence on the complexity of queries, differently from the case of Core XQuery with composition.

Nonrecursive composition-free XQuery is an important class of queries, and indeed, most practical XQueries belong to this class. (For instance, only a handful of the XML Query Use Case queries [15] employ composition.) Composition-free (Core) XQuery is also popular among implementors of limited prototype XQuery engines, e.g. [8]. Our preliminary expressiveness results show that restricting oneself to implementing composition-free Core XQuery does not cause a loss of generality, at least if equality checking is limited to atomic value equality. The expressiveness result also gives a partial explanation for why practical XQueries tend to be composition-free, as observed above. Writing more succinct queries takes an effort, and apparently does not pay off for many queries.

Note that other functional languages such as monad algebra [12] do not seem to have natural "composition-free" fragments that remain expressive.

A major motivation of this work is to define simple but relevant fragments of XQuery suitable for research prototype implementations and theoretical study (see also [4] for another attempt towards this goal). Indeed, composition-free Core XQuery may allow for special, efficient implementation techniques because all variables only range over nodes in the input tree (never over nodes from intermediate query results).

The structure of this paper is as follows. In Section 2, we introduce a clean fragment of nonrecursive XQuery, *Core XQuery*, that will be the language studied in the remainder of the paper. In Section 3 we introduce composition-free Core XQuery. In Section 4, we prove the PSPACE- and NP-completeness results for the complexity of composition-free Core XQuery. Finally, in Section 5, we prove the expressiveness result that composition-free Core XQuery captures full Core XQuery with "child" and atomic equality.

We assume basic complexity classes such as TC_0, NC_1, LOGSPACE, NP, PSPACE, and NEXPTIME known and refer to [6] for the relevant complexity-theoretic background. By TA$[2^{O(n)}, O(n)]$, we denote the class of all problems solvable by alternating Turing machines in linear exponential time with a linear number of alternations (see [6] for definitions). Closure and hardness are under LOGSPACE-reductions for NP, PSPACE, and NEXPTIME and under LOGLIN-reductions (that is, LOGSPACE-reductions whose output is linear) for TA$[2^{O(n)}, O(n)]$.

We use the now standard notions of data, query, and combined complexity introduced by Vardi [13].

2. CORE XQUERY

We consider the fragment of XQuery with abstract syntax

$$
\begin{array}{rcl}
\mathit{query} & ::= & () \mid \langle a\rangle\, \mathit{query} \langle /a\rangle \mid \mathit{query}\ \mathit{query} \\
& \mid & \mathit{var} \mid \mathit{var}/\mathit{axis} :: \nu \\
& \mid & \text{for } \mathit{var} \text{ in } \mathit{query} \text{ return } \mathit{query} \\
& \mid & \text{if } \mathit{cond} \text{ then } \mathit{query} \\
& \mid & (\text{let } \mathit{var} := \langle a\rangle\, \mathit{query} \langle /a\rangle)\ \mathit{query} \\
\mathit{cond} & ::= & \mathit{var} = \mathit{var} \mid \mathit{query}
\end{array}
$$

where a denotes the XML tags, *axis* the XPath axes "child" and "descendant", *var* a set of XQuery variables $\$x$, $\$x_1$, $\$x_2, \ldots, \$y, \$z, \ldots$ with a distinguished *root variable* (which is the unique free variable in the query), and ν a *node test* (either a tag name or "*"). We refer to this fragment as *Core XQuery*, or *XQ* for short.

For simplicity, we will work with pure node-labeled unranked ordered trees, and by atomic values, we will refer to leaves (or equivalently, their labels).

XQuery supports several forms of equality. We will not try to use the same syntax (=, eq, or deep_equal) as in the current standards proposal – it is not clear whether the syntax has stabilized. Throughout this paper, equality is by value (that is, by value as a tree rather than by the *yield* of strings at leaf nodes of the tree). We will write $=_{deep}$ and $=_{atomic}$ for deep and atomic equality, respectively. We will use = for statements that apply to both forms of equality.

Our only other divergence from XQuery syntax is that we assume if-expressions of the form "if ϕ then α" rather than "if ϕ then α else β". Of course, our if-expressions can be considered as a shortcut for "if ϕ then α else ()" and else-branches can be simulated using negation, "if not(ϕ) then β".

We define the semantics of an *XQ* expression α with k free variables using a function $[\![\alpha]\!]_k$ – given in Figure 1 – that takes a k-tuple of trees as input. On input tree t, query Q evaluates to $[\![Q]\!]_1(t)$. The symbol $\uplus$ in Figure 1 denotes list concatenation, l_i the i-th element of list l, $<^t_{doc}$ is the

$$
\begin{array}{rcl}
[\![()]\!]_k(\vec{e}) & := & [\,] \\
[\![\langle a\rangle\alpha\langle/a\rangle]\!]_k(\vec{e}) & := & [\langle a\rangle[\![\alpha]\!]_k(\vec{e})\langle/a\rangle] \\
[\![\alpha\ \beta]\!]_k(\vec{e}) & := & [\![\alpha]\!]_k(\vec{e}) \uplus [\![\beta]\!]_k(\vec{e}) \\
[\![\text{for } \$x_{k+1} \text{ in } \alpha & & \\
\text{return } \beta]\!]_k(\vec{e}) & := & \text{let } l = [\![\alpha]\!]_k(\vec{e}); \\
& & \text{return } \biguplus_{1\le i\le |l|} [\![\beta]\!]_{k+1}(\vec{e}, l_i) \\
[\![(\text{let } \$x_{k+1} := \alpha)\ \beta]\!]_k(\vec{e}) & := & \text{let } l = [\![\alpha]\!]_k(\vec{e}); \\
& & \text{return } [\![\beta]\!]_{k+1}(\vec{e}, l_1) \\
[\![\$x_i]\!]_k(t_1,\ldots,t_k) & := & [t_i] \\
[\![\$x_i/\chi :: \nu]\!]_k(t_1,\ldots,t_k) & := & \text{list of nodes } v \text{ of tree } t_i \text{ s.t.} \\
& & \chi^{t_i}(root^{t_i}, v) \wedge \mathrm{lab}^{t_i}_{\nu}(v) \\
& & \text{in order } <^{t_i}_{doc} \\
[\![\text{if } \phi \text{ then } \alpha]\!]_k(\vec{e}) & := & \text{if } [\![\phi]\!]_k(\vec{e}) \text{ then } [\![\alpha]\!]_k(\vec{e}) \text{ else } [\,] \\
[\![\$x_i = \$x_j]\!]_k(t_1,\ldots,t_k) & := & \text{if } t_i = t_j \text{ then } [\langle yes/\rangle] \text{ else } [\,]
\end{array}
$$

Figure 1: Semantics of Core XQuery.

depth-first left-to-right traversal order through tree t, χ^t is the axis relation χ on t, lab^t_* is true on all nodes of t, and lab^t_a, for a a tag name, is true on those nodes of t labeled a. All XQ queries evaluate to lists of nodes. However, we assume that XQ variables always bind to single nodes rather than lists; our fragment assures this. This semantics is (observationally) consistent with XQuery as currently undergoing standardization through the W3C [14] restricted to Core XQuery.

In our definition of the syntax of Core XQuery, we have been economical with operators introduced. Since conditions are true iff they evaluate to a nonempty collection,

$$
\begin{array}{rcl}
\text{true} & := & \langle a/\rangle \\
\phi \text{ or } \psi & := & \phi\ \psi \\
\phi \text{ and } \psi & := & \text{if } \phi \text{ then } \psi \\
\text{some } \$\text{x in } \alpha \text{ satisfies } \phi & := & \text{for } \$\text{x in } \alpha \text{ return } \phi
\end{array}
$$

Using deep equality, we can define negation,

$$\text{not } \phi := \big(() =_{deep} \text{if } \phi \text{ then } \langle \text{b}/\rangle\big).$$

Conditions "every \$x in α satisfies ϕ" can be defined using "not" and "some". We will use the shortcut $\langle a/\rangle$ for $\langle a\rangle()\langle/a\rangle$. It is clear that

Proposition 2.1. *Let* $\mathbf{X}$ *be a set of operations and axes.*

- *Each* $XQ[=_{deep}, not, every, \mathbf{X}]$ *query can be translated in LOGLIN into an equivalent* $XQ[=_{deep}, \mathbf{X}]$ *query.*
- *Each* $XQ[and, or, some, \mathbf{X}]$ *query can be translated in LOGLIN into an equivalent* $XQ[\mathbf{X}]$ *query.*

Previous results on XQ

The following results on the complexity and expressive power of XQ have been established in [7].

Proposition 2.2 ([7]). *W.r.t. combined complexity,*

- $XQ[=_{deep}, all\ axes]$ *is in EXPSPACE;*
- $XQ[=_{atomic}, all\ axes, not]$ *is in* $TA[2^{O(n)}, O(n)]$*; and*
- $XQ[=_{atomic}, all\ axes]$ *is in NEXPTIME.*

Proposition 2.3 ([7]). *W.r.t. query complexity,*

- $XQ[=_{deep}, child]$ *is* $TA[2^{O(n)}, O(n)]$*-hard;*
- $XQ[=_{atomic}, child, not]$ *is* $TA[2^{O(n)}, O(n)]$*-hard; and*
- $XQ[=_{atomic}, child]$ *is NEXPTIME-hard.*

Proposition 2.4 ([7]). *W.r.t. data complexity,* $XQ[=_{deep}, all\ axes]$ *is*

- *LOGSPACE-complete under* NC_1*-reductions if the XML input is given as a DOM tree and*
- *in* TC_0 *if the XML input is given as a character string.*

It has been shown that many query languages for complex values that were developed earlier, such as nested relational algebra [5], complex value algebra without powerset [1], and monad algebra [12], share the same expressive power. Core XQuery is an interesting fragment of XQuery because it captures precisely this degree of expressiveness, which is commonly deemed "right" for nested, deeply structured data.

Proposition 2.5 ([7]). *XQ[=, child] captures – up to data representation issues – monad algebra on lists [12].*

That is, there are fixed mappings "V2T" (from complex values to trees), "T2V" (from trees to complex values), "M2X" (from monad algebra on lists to XQ), and "X2M" (from XQ to monad algebra on lists), s.t. for XQ query Q and tree T,

$$\mathrm{X2M}(Q)(\mathrm{T2V}(T)) = \mathrm{T2V}(Q(T))$$

and for monad algebra query Q and complex value V,

$$\mathrm{M2X}(Q)(\mathrm{V2T}(V)) = \mathrm{V2T}(Q(V)).$$

The "representation issues" are that the mappings "V2T" and "T2V" are needed. However these are independent from the mappings between the queries.

Proposition 2.5 holds for = being either atomic or deep value equality. From Proposition 2.5 it also follows that $XQ[=, \text{child}]$ is a conservative extension of relational algebra up to representation issues in the spirit of [10], just like monad algebra [12].

3. COMPOSITION-FREE XQ

Composition-free Core XQuery, $XQ^-[\text{not}]$, now is the fragment of Core XQuery obtained by the grammar

$$
\begin{array}{rcl}
query & ::= & () \mid \langle a\rangle query\langle/a\rangle \mid query\ query \\
& \mid & var \mid var/axis :: \nu \\
& \mid & \text{for } var \text{ in } var/axis :: \nu \text{ return } query \\
& \mid & \text{if } cond \text{ then } query \\
cond & ::= & var = var \mid var = \langle a/\rangle \mid \text{true} \\
& \mid & \text{some } var \text{ in } var/axis :: \nu \text{ satisfies } cond \\
& \mid & cond \text{ and } cond \mid cond \text{ or } cond \\
& \mid & \text{not } cond
\end{array}
$$

The keyword "every" can again be obtained from "some" and "not". Testing whether condition $\$x/\chi :: \nu$ (where χ is

an axis and ν is a node test) can be matched is of course possible as "some $\$y$ in $\$x/\chi :: \nu$ satisfies "$\$y = \$y$". *Positive* composition-free Core XQuery XQ^- is again obtained by removing negation "not" from the language.

For our expressiveness proof below, we will use a variant of XQ^- with less syntax, i.e. in which conditions are defined using the usual query operations rather than "some", "and", and "or".

Let $XQ^\sim$ denote the XQ queries

- which do not contain "let"-expressions,
- for which for each expression "for $x in α return β", α is of the form $\$x/\nu$, and
- which in addition support conditions $\$x = \langle a/\rangle$.

$XQ^\sim$ and XQ^- are expressively equivalent.

PROPOSITION 3.1. $XQ^\sim = XQ^-$.

Proof Sketch. $\Rightarrow$: For a mapping from $XQ^\sim$ to XQ^-, we define an appropriate translation function f that we use to rewrite all maximal if-conditions (i.e., conditions of if-expressions that are not subexpressions of if-expressions):

$$\begin{array}{rcl} f(\alpha\ \beta) & := & f(\alpha) \text{ or } f(\beta) \\ f(\text{for \$y in \$x}/\nu \text{ return } \alpha) & := & \text{some \$y in \$x}/\nu \\ & & \text{satisfies f}(\alpha) \\ f(\text{if } \phi \text{ then } \alpha) & := & f(\phi) \text{ and } f(\alpha) \\ f(\text{not } \phi) & := & \text{not } f(\phi) \\ f(\langle a\rangle\alpha\langle/a\rangle) & := & \text{true} \end{array}$$

On all other kinds of expressions, f is the identity.

$\Leftarrow$: For a mapping from XQ^- to $XQ^\sim$, we only need to eliminate "true", "some", "and", and "or" using their definitions from Section 2. □

EXAMPLE 3.2. It is easy to verify that the query

```
<result>
{ for $x in $root/a return
     if not(for $y in $x/b return
               if $y/c then ($y/d $y/e))
     then $x/f }
</result>
```

is $XQ^\sim$. The corresponding XQ^- query is

```
<result>
{ for $x in $root/a return
     if not(some $y in $x/b satisfies
               ($y/c and ($y/d or $y/e)))
     then $x/f }
</result>
```
□

The mappings from the proof of Proposition 3.1 can be implemented efficiently, thus our complexity results established below will hold for both XQ^- and $XQ^\sim$.

4. COMPLEXITY RESULTS FOR XQ⁻

We now provide our complexity characterization of composition-free Core XQuery.

As announced in the introduction, the query evaluation problem for XQ^- is in polynomial space w.r.t. combined complexity.

PROPOSITION 4.1. *$XQ^-[=_{deep}, \text{all axes}, \text{not}]$ is in space $O(|Q| \cdot \log |t|)$, where $|Q|$ is the size of the query and $|t|$ is the size of the data tree.*

Proof Sketch. It is easy to check that by definition of the fragment, XQuery variables always range exclusively over nodes of the input tree. This can be verified by checking the invariant that each variable is introduced using a "for"-statement over a collection defined by an expression $\$x/\nu$, starting at the root node of the input tree.

Thus there is a straightforward algorithm – direct nested-loop based evaluation – for XQ^- queries that only takes memory to store a pointer into the input tree (taking space $\log |t|$) for each of the $O(|Q|)$ variables in the query. □

For the remaining results, we study decision problems and thus Boolean queries. Since valid XML query results have to consist of at least a root node, we say that a Boolean (XQ^-) query $\langle a\rangle\alpha\langle/a\rangle$ returns true iff the root node of the result tree has at least one child.

PROPOSITION 4.2. *$XQ^-[=_{atomic}, \text{child}, \text{not}]$ is PSPACE-hard w.r.t. query complexity.*

Proof Sketch. The problem is PSPACE-hard already with respect to query complexity (i.e., when the input tree is fixed). The proof is a minor variation of the standard proof of the PSPACE-hardness of the relational calculus (cf. [11]), and is by reduction from the Quantified Boolean Formula evaluation problem (QBF). We illustrate it with an example, which should be easy to generalize. Consider the QBF $\forall x \exists y (x \Leftrightarrow y)$, which is true. This formula can be phrased as the query

```
⟨a⟩
{ if every $x in $root/* satisfies
        (some $y in $root/* satisfies
            (not $x="t" or $y="t") and
            (not $y="t" or $x="t"))
then ⟨yes/⟩}
⟨/a⟩
```

over the fixed data tree consisting of a root node with two children, one with string value "t" and the other with string value "f". (Of course, "every $x in Q satisfies ϕ" is the same as "not(some $x in Q satisfies not(ϕ))") □

While negation and universal quantification were redundant in $XQ[=_{deep}]$, and excluding them did not reduce the complexity of the language [7], it is interesting to consider the case of XQ^- without negation.

PROPOSITION 4.3. *$XQ^-[=_{deep}, \text{all axes}]$ is in NP w.r.t. combined complexity.*

Proof Sketch. If the result of the query is to be nonempty, a node has to be written at a certain for-depth k (so the subexpression responsible for the node has up to k free XQuery variables). We can guess the value assignments of these and then check the conditions (this includes axes, node-tests, and if-conditions) along the for-loops up to depth k in polynomial time. (Note that negation only applies to conditions that contain XPath, but no XQuery.) □

PROPOSITION 4.4. *$XQ^-[=_{atomic}, child]$ is NP-hard w.r.t. query complexity.*

Proof Sketch. This follows immediately from the NP-hardness of conjunctive (relational) queries [2], and a proof can be given e.g. by reduction from 3-Colorability: The fixed data tree consists of a root node and three children, which are labeled "red", "green", and "blue", respectively.

Given a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_m\}$ and

$$E = \{\{v_{i(1,1)}, v_{i(1,2)}\}, \ldots, \{v_{i(n,1)}, v_{i(n,2)}\}\}$$

$(1 \leq i(\cdot,\cdot) \leq m)$, we construct the query

```
<result>
{ for $x_1 in $root/* return
   ⋱
      for $x_{m-1} in $root/* return
         for $x_m in $root/* return
         if (not $x_{i(1,1)} =_atomic $x_{i(1,2)}) and ... and
            (not $x_{i(n,1)} =_atomic $x_{i(n,2)}) then
             <yes/>
}
</result>
```

It is easy to verify that indeed this query computes "yes" nodes precisely if G is 3-colorable. Obviously, the query can be computed from G in logarithmic space. □

5. EXPRESSIVENESS OF XQ$^-$

In this final section, we show that surprisingly, for an important case (atomic equality and "child" as the only supported axis), composition-free Core XQuery is actually just as expressive as full Core XQuery. This is true even though XQ^- is in PSPACE and XQ is hard for TA$[2^{O(n)}, O(n)]$. Thus under commonly-held complexity theoretic assumptions, XQ is exponentially more succinct than XQ^-.

We use the shortcut $(\langle a\rangle\alpha\langle/a\rangle)/\chi{::}\nu$ for $\$x/\chi{::}\nu$ such that $\$x$ has been defined using "let" as $(\langle a\rangle\alpha\langle/a\rangle)$. Below, "dos" is a shortcut for the "descendant-or-self" axis; it will be redundant because $\$x/\text{dos}{::}\nu$ is equivalent to

$$(\text{if } \$x/\text{self}{::}\nu \text{ then } \$x)\ \$x//\nu.$$

LEMMA 5.1. *Let a be a label, χ an axis, ν a nodetest, and α an $XQ^\sim[=_{atomic}, child, descendant, self, dos, not]$ expression. Then there is an $XQ^\sim[=_{atomic}, child, descendant, self, dos, not]$ expression equivalent to $(\langle a\rangle\alpha\langle/a\rangle)/\chi{::}\nu$.*

Proof Sketch. Rules to rewrite each such expression

$$(\langle a\rangle\alpha\langle/a\rangle)/\chi{::}\nu$$

into an equivalent $XQ^\sim[=_{atomic}, \text{child, descendant, self, not}]$ expression are easy to specify:

$$\begin{array}{rcl}
(\langle a\rangle\ \alpha\ \langle/a\rangle)/\nu & \vdash & \alpha/\text{self}{::}\nu \\
(\langle a\rangle\ \alpha\ \langle/a\rangle)/\text{self}{::}\text{b} & \vdash & () \\
(\langle a\rangle\ \alpha\ \langle/a\rangle)/\text{self}{::}\text{a} & \vdash & \langle a\rangle\ \alpha\ \langle/a\rangle \\
(\langle b\rangle\ \alpha\ \langle/b\rangle)/\text{self}{::}* & \vdash & \langle b\rangle\ \alpha\ \langle/b\rangle \\
(\langle a\rangle\ \alpha\ \langle/a\rangle)//\nu & \vdash & \alpha/\text{dos}{::}\nu \\
(\langle a\rangle\ \alpha\ \langle/a\rangle)/\text{dos}{::}* & \vdash & \langle a\rangle\ \alpha\ \langle/a\rangle\ (\alpha//*) \\
(\langle a\rangle\ \alpha\ \langle/a\rangle)/\text{dos}{::}a & \vdash & \langle a\rangle\ \alpha\ \langle/a\rangle\ (\alpha//a) \\
(\langle a\rangle\ \alpha\ \langle/a\rangle)/\text{dos}{::}b & \vdash & \alpha//b \\
()/\chi{::}\nu & \vdash & ()
\end{array}$$

$$\begin{array}{rcl}
(\alpha\ \beta)/\chi{::}\nu & \vdash & (\alpha/\chi{::}\nu)\ (\beta/\chi{::}\nu) \\
(\text{for \$x in } \alpha \text{ return } \beta)/\chi{::}\nu & \vdash & \text{for \$x in } \alpha \\
 & & \text{return } (\beta/\chi{::}\nu) \\
(\text{if } \phi \text{ then } \alpha)/\chi{::}\nu & \vdash & \text{if } \phi \text{ then } (\alpha/\chi{::}\nu) \\
(\$x/\chi{::}\nu)/\chi'{::}\nu' & \vdash & \text{for } \$y \text{ in } \$x/\chi{::}\nu \\
 & & \text{return } \$y/\chi'{::}\nu'
\end{array}$$

(Note that $(\$x/\chi{::}\nu)/\chi'{::}\nu'$ in the final rule is really equivalent to the for-expression on the right-hand side of that rule, and is in general not equivalent to $\$x/\chi{::}\nu/\chi'{::}\nu'$, as the former may produce duplicates if both χ and χ' are "descendant".) □

THEOREM 5.2. *$XQ^\sim[=_{atomic}, child, desc, self, not]$ captures the $XQ[=_{atomic}, child, desc, self, not]$ queries.*

Proof Sketch. We first replace each expression of the form "(let $\$x := \langle a\rangle\alpha\langle/a\rangle$) β" by an expression $\beta' := \beta[\$x \Rightarrow \langle a\rangle\alpha\langle/a\rangle]$ obtained by substituting each occurrence of variable $\$x$ in β by $\langle a\rangle\alpha\langle/a\rangle$.

We now need to consider where such a replacement of a variable $\$x$ by an expression $\langle a\rangle\alpha\langle/a\rangle$ can occur:

1. Inside an equality $\$x =_{atomic} \alpha$ (with α either a variable or a constant $\langle b/\rangle$).

   To rewrite $\$x$ with $\langle a\rangle\alpha\langle/a\rangle$, we may assume that α is (); otherwise, we could not type $\langle a\rangle\alpha\langle/a\rangle$ to be an atomic value. Thus we obtain $\langle a/\rangle =_{atomic} \alpha$, which is $XQ^\sim$. Conditions $\langle a/\rangle =_{atomic} \langle a/\rangle$ and $\langle a/\rangle =_{atomic} \langle b/\rangle$ are rewritten into "true" and "not(true)", respectively.

2. Inside an expression $\$x$ or $\$x/\chi{::}\nu$ (either in the "in"-expression of a for-loop or as an expression constructing "output").

   Here rewriting may lead to expressions of the form $(\langle a\rangle\alpha\langle/a\rangle)/\chi{::}\nu$, which is not XQ syntax. We can eliminate such expressions using Lemma 5.1.

Now the query obtained is already an $XQ^\sim$ query if in all expressions "for $\$x$ in α return β", α is of the form $\$z$ or $\$z/\chi{::}\nu$. Otherwise, we apply the rewrite rules from Figure 2. This may again produce expressions $(\langle a\rangle\alpha\langle/a\rangle)/\chi{::}\nu$, by rule (2). We eliminate such cases again using Lemma 5.1.

It can be verified that the rewrite system thus specified indeed maps any $XQ[=_{atomic}, \text{child, desc, self, not}]$ query to an equivalent $XQ^\sim[=_{atomic}, \text{child, desc, self, not}]$ query. An example mapping to $XQ^\sim$ illustrating our rewrite system is given in Figure 3. □

Acknowledgments

I thank Dan Olteanu and Stefanie Scherzinger for their comments on an earlier version of the paper.

6. REFERENCES

[1] S. Abiteboul and C. Beeri. "The Power of Languages for the Manipulation of Complex Values". *VLDB J.*, **4**(4):727–794, 1995.

[2] A. K. Chandra and P. M. Merlin. "Optimal Implementation of Conjunctive Queries in Relational Data Bases". In *Conference Record of the Ninth Annual ACM Symposium on Theory of Computing (STOC'77)*, pages 77–90, Boulder, CO, USA, May 1977.

$$
\begin{aligned}
\text{for \$x in () return } \alpha &\vdash \text{()} && (1)\\
\text{for \$x in } (\langle a\rangle\ \alpha\ \langle /a\rangle) \text{ return } \beta &\vdash \beta[\$x \Rightarrow (\langle a\rangle\ \alpha\ \langle /a\rangle)] && (2)\\
\text{for \$x in } (\alpha\ \beta) \text{ return } \gamma &\vdash (\text{for \$x in } \alpha \text{ return } \gamma)\ (\text{for \$x in } \beta \text{ return } \gamma) && (3)\\
\text{for \$y in (for \$x in } \alpha \text{ return } \beta) \text{ return } \gamma &\vdash \text{for \$x in } \alpha \text{ return for \$y in } \beta \text{ return } \gamma && (4)\\
\text{for \$x in (if } \phi \text{ then } \alpha) \text{ return } \beta &\vdash \text{for \$x in } \alpha \text{ return if } \phi \text{ then } \beta && (5)\\
\text{for \$y in \$x return } \alpha &\vdash \alpha[\$y \Rightarrow \$x] && (6)
\end{aligned}
$$

Figure 2: Rewrite rules for translating for-expressions to $XQ^{\sim}$.

$$
\begin{aligned}
(\text{let } \$x := \langle a\rangle\{\text{ for } \$w \text{ in \$root/* return } \langle b\rangle\{\$w\}\langle /b\rangle\ \}\langle /a\rangle) \text{ for } \$y \text{ in } \$x/b \text{ return } \$y/* &\overset{elim.let}{\vdash}\\
\text{for } \$y \text{ in } (\langle a\rangle\{\text{ for } \$w \text{ in \$root/* return } (\langle b\rangle\{\$w\}\langle /b\rangle)\ \}\langle /a\rangle)/b \text{ return } \$y/* &\overset{\text{Lem. 5.1}}{\vdash}\\
\text{for } \$y \text{ in (for } \$w \text{ in \$root/* return } (\langle b\rangle\{\$w\}\langle /b\rangle)) \text{ return } \$y/* &\overset{4}{\vdash}\\
\text{for } \$w \text{ in \$root/* return for } \$y \text{ in } (\langle b\rangle\{\$w\}\langle /b\rangle) \text{ return } \$y/* &\overset{2}{\vdash}\\
\text{for } \$w \text{ in \$root/* return } (\langle b\rangle\{\$w\}\langle /b\rangle)/* &\overset{\text{Lem. 5.1}}{\vdash}\\
\text{for } \$w \text{ in \$root/* return } \$w &
\end{aligned}
$$

Figure 3: Example rewriting.

[3] G. Gottlob, C. Koch, and R. Pichler. "Efficient Algorithms for Processing XPath Queries". In *Proc. VLDB 2002*, pages 95–106, Hong Kong, China, 2002.

[4] J. Hidders, J. Paredaens, R. Verkammen, and S. Demeyer. "A Light but Formal Introduction to XQuery". In *Proc. XSYM*, pages 5–20, 2004.

[5] G. Jaeschke and H.-J. Schek. "Remarks on the Algebra of Non First Normal Form Relations". In *Proc. PODS'82*, pages 124–138, 1982.

[6] D. S. Johnson. "A Catalog of Complexity Classes". In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 1, chapter 2, pages 67–161. Elsevier Science Publishers B.V., 1990.

[7] C. Koch. "On the Complexity of Nonrecursive XQuery and Functional Query Languages on Complex Values". In *Proc. PODS'05*, 2005.

[8] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier. "Schema-based Scheduling of Event Processors and Buffer Minimization for Queries on Structured Data Streams". In *Proc. VLDB 2004*, Toronto, Canada, 2004.

[9] A. Marian and J. Siméon. "Projecting XML Documents". In *Proc. VLDB 2003*, pages 213–224, 2003.

[10] J. Paredaens and D. Van Gucht. "Possibilities and Limitations of Using Flat Operators in Nested Algebra Expressions". In *Proc. PODS*, pages 29–38, 1988.

[11] L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory*. PhD thesis, Dept. Electrical Engineering, MIT, Cambridge, Mass., USA, 1974.

[12] V. Tannen, P. Buneman, and L. Wong. "Naturally Embedded Query Languages". In *Proc. of the 4th International Conference on Database Theory (ICDT)*, pages 140–154, 1992.

[13] M. Y. Vardi. "The Complexity of Relational Query Languages". In *Proc. 14th Annual ACM Symposium on Theory of Computing (STOC'82)*, pages 137–146, San Francisco, CA USA, May 1982.

[14] World Wide Web Consortium. "XQuery 1.0 and XPath 2.0 Formal Semantics. W3C Working Draft (Aug. 16th 2002), 2002. `http://www.w3.org/TR/query-algebra/`.

[15] "XML Query Use Cases. W3C Working Draft 02 May 2003", 2003. http://www.w3.org/TR/xmlquery-use-cases/.