Querying XML with FLWR Expressions in $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$: On the Utility of Interval Based Encoding

David Toman and Grant Weddell
Department of Computer Science
University of Waterloo, Canada
{david,gweddell}@uwaterloo.ca

November 16, 2001

Abstract

In addition to XPath queries and XML patterns, the most recent draft by a W3C group for an XML query language includes so-called FLWR expressions in a core fragment that serves as the basic means for querying and transforming XML documents. In this paper, we compare transformations of XML documents that can be defined in this core language to those that can be defined by first order queries with safe arithmetic, and with the possible use of counting, over interval based relational encodings of XML documents. Such encodings are important because of their generality, they work for arbitrary XML documents, and because they and their associated decodings to XML documents can be computed efficiently. We also consider how additional kinds of data computed by an encoding process and knowledge of document structure can lead to improved performance for queries.

1 Introduction

We consider a confluence of two trends relating to XML transformation. The first relates to the most recent draft by a W3C group for an XML query language called XQuery [4] in which the authors have included FLWR expressions in a core fragment to serve as the basic means for transforming XML documents. Such expressions closely resemble a similar feature in an earlier language called QUILT [5]. The second trend relates to a growing interest in computing such transformations in a manner roughly illustrated by the commuting diagram in Figure 1. In particular, (1) relations that *represent* one or more XML documents can be computed by an *encoding* procedure (this essentially amounts to simple parsing of the input XML string), (2) a mapping function Q is applied to an XML query e to obtain a relational query Q_e over the relations, and finally (3) a decoding procedure (a simple SQL *order by* query) is applied to the result of Q_e to obtain the XML document that would have been computed by e. This approach has considerable appeal since it incorporates existing relational technology in the transformation process. Consequently, there is a potential for reuse of algorithmic capabilities manifested in current relational technology.

In this paper, we develop a technique that maps queries over XML documents that can be defined in this core fragment of XQuery, including FLWR expressions, to those that can be defined by first order queries with safe natural number arithmetic and possibly with counting, denoted $\mathsf{FO}_{\mathbb{N}}$ and $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$, respectively, using the approach suggested by Figure 1. The queries we generate are then executed over an interval based representation of XML documents that uses *natural numbers* to represent endpoints of intervals and interval containment to capture and compute ancestry in the rooted ordered forests underlying the XML data model [7]. Other researchers have also used interval based representations to compute various subsets of XQuery. In particular, [2, 14] consider their use in the computation of containment queries, and [8, 12] consider the

$$
\begin{tikzcd}
\text{XML} \arrow[r, "e\in\text{XQuery}"] \arrow[d, shift right, "\text{encodes}"'] & \text{XML} \\
\text{Relation(s)} \arrow[r, "Q_e\in\text{FO+count}"] \arrow[u, shift right, "\text{represents}"'] & \text{Relation(s)} \arrow[u, "\text{represents}"']
\end{tikzcd}
$$

Figure 1: Mapping XML Queries to Relational Queries.

problem of mapping queries based on dialects of XPath [6] to SQL queries over such encodings. So far as we are aware, our translation of the FLWR fragment of XQuery is the first to account for the following features:

- The translation is fully compositional (cf. Figure 3); in particular, the translation correctly maps *arbitrarily nested* FLWR expressions to relational queries,
- The resulting queries are in $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$, and, in many cases, even in $\mathsf{FO}_{\mathbb{N}}$. This guarantees low data complexity, and
- Beyond the analytical complexity bounds, the generated queries can be efficiently evaluated with only minor modifications of operators present in standard relational engines.

In particular, our approach overcomes common objections to using relational technology to querying and transforming XML documents, namely problems connected with (1) nesting in XML documents vs. flat relations, (2) recursive nature of XML operations (e.g., the *ancestor/descendant* like operations) vs. the first-order nature of relational queries, and (3) the requirement of preservation of *document order* and the need for *sorting within queries* vs. the set based nature of the relational model.

In addition, the interval based encoding, while completely general, is rather compact: we need $O(n \log n)$ bits for document of size n; this provides a much better space utilization that encodings based on *transitive closure* [9], that in the worst case may need $O(n^2)$ space.

By far the most difficult part of the translation proposed in this paper is the problem of mapping FLWR expressions to $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$ queries. To see some of this difficulty, consider the following XQuery for computing *a sorted list of authors for each paper occurring in a document.*

```
for $p in doc//paper
   return <paper>
      $p/title
      sort($p//author)
   </paper>
```

Note in particular that the sort operation prevents an unfolding of the nested FLWR expression. A translation of this query to $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$ must therefore encode computations of genuine nested *and ordered* results. The *iterated ordered semantics* of XQuery also prevents other unfolding rewrites of XQuery FLWR expressions that would otherwise simplify the translation process. One such rewrite, used in [9] for example, is to rewrite the query

```
for $x in doc, $y in (for $z in doc return doc/) return doc
```

with the query

```
for $x in doc, for $z in doc return doc
```

In this case, the queries return different results if the document root contains more than one child.

The rest of the paper is organized as follows. Section 2 gives a brief overview of the XML data model (simplified for the purposes of this paper), lists a sample of *built-in* functions commonly present in XML query languages, and defines the syntax and semantics of our XQuery's FLWR-like core. Section 3 then introduces XML relations, a relational encoding of XML documents based on intervals. The translation from this language to $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$ is presented in Section 4. The section proceeds in two steps: first, we give examples of how to translate a number of built-in XML functions[1]. Second, we define a formal translation of FLWR-like expressions to $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$ queries. Additional implementation issues related to interval based encodings are then outlined in Section 5. We conclude with suggestions for directions of future research, and with a list of several open questions connected with this approach.

2 XML Data and Query Primer

In this section, we define a simple model for XML data that will suffice for our illustrative purposes, and define an XQuery-like query language for this model that consists of: (1) a built-in collection of *XML functions*, denoted XFn, and (2) XQuery's FLWR expressions. A subset of XFn, the so-called *constructor* functions, correspond to XML patterns in XQuery, and (although the details are beyond the scope of this paper) arbitrary XPath queries can be straightforwardly mapped to a combination of XFn and FLWR expressions in our language.

2.1 XML Data

As is common with most papers on XML, we interpret an XML document or a document collection as an ordered forest of rooted node-labeled ordered trees.

Definition 2.1 (XML Forests and Trees) *Let* String *be a set of strings. We define the set* XF *of XML forests inductively by*

$$\mathsf{XF} \quad = \quad [\,] \;\mid\; [\mathsf{XNode}(s,x)] \;\mid\; x@y,$$

where s is in String*, x and y are in* XF*, $[\,]$ denotes an empty forest*[2]*, $[\mathsf{XNode}(s,x)]$ denotes a forest containing a single tree with a root labeled s and an ordered forest x, and $x@y$ denotes a concatenation of two ordered forests.*

Our formulation of XML is very simple with no explicit accounting of node identity, of *element*, *attribute* or *text* node types, or of "don't care" child node order. Such features, however, can be easily added by additional encoding conventions that relate either to node labeling or to subtree patterns. A text leaf node with CDATA "`abc`", for example, might be encoded using the label "`text:abc`". Similar approach can be taken to represent attributes, etc.

2.2 Operations on XML Forests

We restrict our attention to a representative sample of *primitive* functions on XML forests that comprise XFn. Although the set of functions that we have chosen is by no means *complete* (as witnessed by, e.g., the XQuery document [4]), our definition of XFn is sufficient to cover a wide variety of situations. In general, we allow functions to be composed in a way that satisfies the usual type-compatibility rules. We assume the usual operations on primitive types (e.g., conjunction, negation, etc., for the Bool type). Also, as a notational convention for the remainder of the paper, we use e, possibly subscripted, to denote expressions returning elements of XF.

2.3 Boolean Functions and Notions of Equality in XML

XML query languages commonly support three kinds of equality (and analogously, comparison) tests.

[1]These translations are introduced informally; a proof of their correctness would require us to introduce formal semantics for the built-in functions.

[2]We employ common list notation to represent ordered forests.

Constructors:	$[\,] : \mathsf{XF}$	the empty forest constructor
	$\mathsf{XNode} : \mathsf{String} \times \mathsf{XF} \to \mathsf{XF}$	the singleton constructor
	$@ : \mathsf{XF} \times \mathsf{XF} \to \mathsf{XF}$	the concatenation operator
Horizontal Operations:	$\mathsf{head} : \mathsf{XF} \to \mathsf{XF}$	first element of a forest
	$\mathsf{tail} : \mathsf{XF} \to \mathsf{XF}$	all but the first element of the forest
	$\mathsf{reverse} : \mathsf{XF} \to \mathsf{XF}$	the forest in reverse order (top-level only)
	$\mathsf{select}_s : \mathsf{XF} \to \mathsf{XF}$	subforest with trees labeled s in their root
	$\mathsf{distinct}_= : \mathsf{XF} \to \mathsf{XF}$	subforest of =-distinct trees (1st preserved)
	$\mathsf{sort}_< : \mathsf{XF} \to \mathsf{XF}$	a forest ordered by $<$
Vertical Operations:	$\mathsf{roots} : \mathsf{XF} \to \mathsf{XF}$	a forest of root nodes
	$\mathsf{children} : \mathsf{XF} \to \mathsf{XF}$	a forest of all children in original order
	$\mathsf{subtreesdfs} : \mathsf{XF} \to \mathsf{XF}$	a forest of all subtrees in DFS order
Boolean Operations:	$\mathsf{equal} : \mathsf{XF} \times \mathsf{XF} \to \mathsf{Bool}$	test for structural equality
	$\mathsf{less} : \mathsf{XF} \times \mathsf{XF} \to \mathsf{Bool}$	structural ordering
	$\mathsf{empty} : \mathsf{XF} \to \mathsf{Bool}$	test for emptiness

Figure 2: SAMPLE BASIC OPERATIONS ON XML FORESTS.

1. *Label Equality*: Two nodes are equal if their labels are the same.
2. *Structural Equality*: Two nodes are equal if their labels are equal, each has the same number of children, and each pair of corresponding children are in turn structurally equal.
3. *Node ID Equality*: Two nodes are equal if they are the same node in a particular document.

The distinct and sort operations are thus indexed by the appropriate equality/order boolean function (that, in general, may be different from the equal and the less functions).

2.4 FLWR-like Queries

The functions on XML forests defined in Section 2.2 are used to define basic operations. They can be combined into more complex expressions using the language[3] defined in Figure 3. The syntax allows composing builtin functions, definitions of local variables, filtering by boolean conditions, and iteration over members of a forest. The semantics of these expressions is defined inductively with respect to an *environment* E that provides XML values for free variables. Without loss of generality, the syntax does not explicitly account for constant names for XML documents (e.g., the *input* documents). These can be provided as variable bindings in an *initial environment*.

3 Interval Encoding of XML

We now turn our attention to the main goal of this paper—to representing XML data as instances of relations and XML queries as $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$ queries over the instances.

Definition 3.1 (An Interval Encoding) *Let $x \in \mathsf{XF}$ be an XML forest. We define an instance of a ternary relation $X \subseteq \mathsf{String} \times \mathsf{Nat} \times \mathsf{Nat}$ to contain a tuple (s, l, r) for every node $\mathsf{XNode}(s, y)$ in the forest x. The values l and r are constrained as follows:*

- *$\mathsf{XNode}(s_1, y_1)$ is a parent of $\mathsf{XNode}(s_2, y_2)$ in x if and only if the corresponding tuples $(s_1, l_1, r_1) \in X$, and $(s_2, l_2, r_2) \in X$ representing these two nodes satisfy $l_1 < l_2$ and $r_2 < r_1$ and*
- *if the node $\mathsf{XNode}(s_1, y_1)$ is a left sibling of the node $\mathsf{XNode}(s_2, y_2)$ in x then the corresponding tuples $(s_1, l_1, r_1) \in X$ and $(s_2, l_2, r_2) \in X$ satisfy $r_1 < l_2$.*

[3] While we have simplified the syntax and fixed the semantics using a denotational semantics-style definition of the $[\![.]\!]$ map, this language still captures the intricate nuances present in the XQuery's FLWR expressions.

$$e ::= x \mid \mathsf{XFn}(e_1,\ldots,e_k) \mid \mathsf{let}\, x = e_1 \,\mathsf{in}\, e_2 \mid \mathsf{where}\, \varphi \,\mathsf{return}\, e \mid \mathsf{for}\, x \in e_1 \,\mathsf{do}\, e_2$$

$$\begin{array}{rcl}
[\![x]\!] E & = & E(x) \\
[\![\mathsf{XFn}(e_1,\ldots,e_k)]\!] E & = & [\![\mathsf{XFn}]\!]([\![e_1]\!] E,\ldots,[\![e_k]\!] E) \\
[\![\mathsf{let}\, x = e_1 \,\mathsf{in}\, e_2]\!] E & = & [\![e_2]\!](E[x := ([\![e_1]\!] E)]) \\
[\![\mathsf{where}\, \varphi \,\mathsf{return}\, e]\!] E & = & \text{if } [\![\varphi]\!] E \text{ then } [\![e]\!] E \text{ else } [\,] \\
[\![\mathsf{for}\, x \in e_1 \,\mathsf{do}\, e_2]\!] E & = & \mathsf{let}\ [v_1,\ldots,v_k] = [\![e_1]\!] E \\
 & & \quad \mathsf{in}\ [\![e_2]\!](E[x := v_1])@\ldots@[\![e_2]\!](E[x := v_k])
\end{array}$$

Figure 3: Syntax and Semantics of FLWR-like Expressions.

We say that X represents (is a XML relation *for) x.*

We associate a value $w \in \mathsf{Nat}$ that satisfies the condition $w > \max\{r : (s,l,r) \in X\}$ with a fixed representation X of an XML forest x. We call w the width of *(the representation of) x.*

An XML relation X is tight *if the values $r_s - l_s$ are minimal and* pointed *if the minimal value l_s is equal to 0.*

Lemma 3.2 *For a fixed representation X of an XML forest x the width of x can be defined by a $\mathsf{FO}_{\mathbb{N}}$ query on X.*

In the remainder of the paper, we use lower-case identifiers to denote XML documents and their upper-case counterparts to denote their relational encodings.

4 Translation of XML Queries to $\mathsf{FO}_{\mathbb{N}}$ or $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$

We present our main results in this section; essentially, we show that most constructs commonly adopted in various XML query languages can be equivalently expressed in $\mathsf{FO}_{\mathbb{N}}$ or $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$ over XML relations.

4.1 Translation of Built-in Functions

We start with the translations of the builtin XML operations (cf. Figure 2) to $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$ queries.

Definition 4.1 *Let $\mathsf{XFn}(x_1,\ldots,x_k)$ be a builtin function on XML forests. We say that* a $\mathsf{FO}_{\mathbb{N}}$ ($\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$) query Q_{XFn} over the schema $X_1,\ldots,X_k$ represents the function XFn *if the result of evaluating $Q_{\mathsf{XFn}}(X_1,\ldots,X_k)$ represents $\mathsf{XFn}(x_1,\ldots,x_k)$ in any instance where X_i represents x_i (for $0 < i \leq k$).*

In addition, let $w_{\mathsf{XFn}} : \mathsf{Nat}^k \to \mathsf{Nat}$ be a function for which $w_{Q_{\mathsf{XFn}}(X_1,\ldots,X_k)} \leq w_{\mathsf{XFn}}(w_{x_1},\ldots,w_{x_k})$ whenever w_{x_i} is the width of the representation of the XML tree x_i.

Note that the query Q_{XFn} can be used as a query template that depends on the particular names of the input XML relations. This way we can freely compose such queries in the same way the XFn functions are composed. Boolean-valued functions are mapped to closed $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$ queries analogously.

Theorem 4.2 (Translations of Builtin Functions)

1. *Let* $\mathsf{XFn} \in \{[\,], \mathsf{XNode}, @, \mathsf{head}, \mathsf{tail}, \mathsf{select}_s, \mathsf{roots}, \mathsf{children}, \mathsf{subtreesdfs}\}$. *Then there is a $\mathsf{FO}_{\mathbb{N}}$ query Q_{XFn} that represents XFn. Also, there is a closed $\mathsf{FO}_{\mathbb{N}}$ query Q_{empty} that represents* empty.

2. *Let* $\mathsf{XFn} \in \{\mathsf{equal}, \mathsf{less}, \mathsf{sort}\}$*. Then there is a* $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$ *query* Q_{XFn} *that represents* XFn*. However, there is no* $\mathsf{FO}_{\mathbb{N}}$ *query that represents* XFn*.*

3. *Let* $\mathsf{XFn} \in \{\mathsf{distinct}\}$*. Then there is a query* Q_{XFn} *that represents* XFn*.* Q_{XFn} *is a* $\mathsf{FO}_{\mathbb{N}}$ *query if the comparison for equality can be expressed in* $\mathsf{FO}_{\mathbb{N}}$*; and a* $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$ *query otherwise.*

In addition, for each of the above functions XFn *the function* w_{XFn} *is a polynomial function.*

Here are several examples of such translations; full listing is provided in Appendix A.

Example 4.3

$$\begin{array}{rcl}
Q_{@}(X_1, X_2) & = & X_1 \cup \{(s, l + w_{x_1}, r + w_{x_1}) : (s, l, r) \in X_2\} \\
Q_{\mathsf{roots}}(X) & = & \{(s, l, r) : (s, l, r) \in X \wedge \neg\exists s', l', r'.(s', l', r') \in X \wedge l' < l \wedge r < r'\} \\
Q_{\mathsf{subtreesdfs}}(X) & = & \{(s, l + w_x l', r + w_x l') : (s, l, r) \in X \wedge (s', l', r') \in X \wedge l' \leq l \wedge r \leq r'\} \\
Q_{\mathsf{reverse}}(X) & = & \{(s, l - l' + (w_x - r'), r - l' + (w_x - r')) : (s, l, r) \in X \wedge \\
 & & \qquad (s', l', r') \in Q_{\mathsf{roots}}(X) \wedge l' \leq l \wedge r \leq r'\} \\
Q_{\mathsf{empty}}(X) & = & (\neg\exists s, l, r.(s, l, r) \in X)
\end{array}$$

where $w_{@} = w_{x_1} + w_{x_2}$, $w_{\mathsf{reverse}} = w_x$, $w_{\mathsf{roots}} = w_x$, and $w_{\mathsf{subtrees}} = w_x^2$.

This example also shows that even operations traditionally out of the reach of $\mathsf{FO}_{\mathbb{N}}$ queries, e.g., the $\mathsf{subtreesdfs}$ operation that creates a forest of all subtrees of its input in DFS order, can be expressed on top of the interval encoding.

4.2 FLWR Translation

The semantics of FLWR expressions is defined with respect to an *environment* E that maps variable names to XML forests (cf. Figure 3). In the case of "$\mathsf{for}\, x \in e_1 \,\mathsf{do}\, e_2$" expressions, the semantic equation defines an iterator over a sequence of such environments; the result is then the concatenation of the results from the individual iterations. To simulate this action by a fixed relational query, we need to *map* a translation of a (primitive) operation and in turn of an expression to a query over the representation of *all environments* that participate in evaluation of the expression.

Definition 4.4 *Let* $[E_1, \ldots, E_n]$ *be a sequence of environments over the variables* $x_1, \ldots, x_m$ *of the form* $E_i = \langle x_1 = t_1^i, \ldots, x_m = t_m^i \rangle$*. We say that an instance of a relational schema* $I, T_{x_1}, \ldots, T_{x_m}$*, where* $I \subseteq \mathsf{Nat}$ *and* T_{x_i} *is an XML relation,* represents $[E_1, \ldots, E_n]$ *if*

- $I = \{i_1, \ldots, i_n\}$ *such that* $i_k < i_l$ *whenever* $k < l$ *and*
- $T_{x_j} = \bigcup_{k=1}^{n} \{(s, l + i_k w_{x_i}, r + i_k w_{x_i}) : (s, l, r) \in X_j^k\}$ *where* X_j^k *represents* t_j^k

for $w_{x_j} = max\{w : w \text{ width of } X_j^k\}$*. We call* w_{x_j} *the* width *of (the representation of)* x_j*.*

In essence, we have created a representation that *encodes* a sequence of tuples of XML forests representing the values for variables together with an *index set* that tells us how many such tuples there are and in what range the values bound to variables in a particular tuple occupy in the overall interval encoding. This construction is used to simulate sequences of environments for FLWR expressions. In the rest of the section we assume that $I, T_{x_1}, \ldots, T_{x_m}$ represents the sequence of environments $[E_1, \ldots, E_n]$ as above.

Translation of Function Applications. For XFn a k-ary function on XF represented by Q_{XFn} we define the query

$$\begin{aligned}T_{\mathsf{XFn}(x_1,\ldots,x_m)} = \{(s, l + iw_{\mathsf{XFn}}(w_{x_1},\ldots,w_{x_m}), r + iw_{\mathsf{XFn}}(w_{x_1},\ldots,w_{x_m})) :\\ (s,l,r) \in Q_{\mathsf{XFn}}(X_1,\ldots,X_m) \wedge i \in I\}\end{aligned}$$

where $X_k = \{(s, l - iw_{x_k}, r - iw_{x_k}) : (s,l,r) \in T_{x_k} \wedge iw_{x_k} \le l \wedge r < (i+1)w_{x_k}\}$. The pair of instances $I, T_{\mathsf{XFn}(x_1,\ldots,x_m)}$ then represents the sequence $[\mathsf{XFn}(t_1^1,\ldots,t_m^1),\ldots,\mathsf{XFn}(t_1^n,\ldots,t_m^n)]$. In addition, the formula defining $T_{\mathsf{XFn}(x_1,\ldots,x_m)}$ is in $\mathsf{FO}_{\mathbb{N}}$ (modulo the definition of Q_{XFn}). The construction immediately extends to arbitrary FLWR expressions e built from variables $x_1,\ldots,x_m$ and builtin functions; this yields a query T_e such that I, T_e represents $[[\![e]\!] E_1,\ldots,[\![e]\!] E_n]$.

Translation of the Assignment Statement. The above construction also provides the basics for the translation of the $\mathsf{let}\, x = e_1 \,\mathsf{in}\, e_2$ assignment expressions. However, such an expression *changes the environment* (by adding the binding for $x = e_1$ before executing e_2). Therefore we first *extend* the relational representation of the sequence of environments as follows:

$$I' = I, T'_x = T_{e_1}, \text{ and } T'_{x_1} = T_{x_1},\ldots,T'_{x_m} = T_{x_m}$$

We then apply the translation of e_2 to this new (primed[4]) representation $I', T'_{x_1},\ldots,T'_{x_m}, T'_x$ obtaining T'_{e_2}. Then we define the result as $T_{\mathsf{let}\, x=e_1 \,\mathsf{in}\, e_2} = T'_{e_2}$ and $w_{\mathsf{let}\, x=e_1 \,\mathsf{in}\, e_2} = w_{e_2}$.

Translation of the Conditional Statement. Similarly to the assignment we first define a representation $I', T'_{x_1},\ldots,T'_{x_m}$ of a subsequence of $[E_1,\ldots,E_n]$ in which each environment satisfies φ as follows:

$$\begin{aligned}I' &= \{i : i \in I \wedge Q_\varphi(X_1,\ldots,X_m)\}\\ T'_{x_k} &= \{(s,l,r) : i \in I' \wedge (s,l,r) \in T_{x_k} \wedge iw_{x_k} \le l \wedge r < (i+1)w_{x_k}\}\end{aligned}$$

where $X_k = \{(s, l - iw_{x_k}, r - iw_{x_k}) : (s,l,r) \in T_{x_k} \wedge iw_{x_k} \le l \wedge r < (i+1)w_{x_k}\}$. We use this this representation to define T'_e. Then $T_{\mathsf{where}\,\varphi\,\mathsf{return}\, e} = T'_e$ and $w_{\mathsf{where}\,\varphi\,\mathsf{return}\, e} = w_e$.

Translation of the Iterator. We define $I', T'_{x_1},\ldots,T'_{x_m}, T'_x$ to represent the sequence of environments $[E_1[x = t_1^{l_1^1}],\ldots,E_1[x = t_1^{l_1^{k_1}}],\ldots,E_n[x = t_n^{l_n^1}],\ldots,E_1[x = t_n^{l_n^{k_n}}]$ for $[t_i^{l_i^1},\ldots,t_i^{l_i^{k_i}}] = [\![e_1]\!] E_i$, $0 < i \le n$ required by the semantics of the $\mathsf{for}\, x \in e_1 \,\mathsf{do}\, e_2$ iterator expression. The representation is defined as follows:

$$\begin{aligned}I' &= \{iw_{e_1} + l_r : i \in I \wedge (l_r, r_r) \in R_{e_1}(i)\}\\ T'_x &= \{(s, l - iw_{e_1} + (iw_{e_1} + l_R)w_{e_1}, r - iw_{e_1} + (iw_{e_1} + l_R)w_{e_1}) :\\ &\qquad i \in I \wedge (l_R, r_R) \in R_{e_1}(i) \wedge (s,l,r) \in T_{e_1} \wedge l_R \le l \wedge r \le r_R\}\\ T'_{x_k} &= \{(s, l - iw_{x_k} + (iw_{e_1} + l_R)w_{x_k}, r - iw_{x_k} + (iw_{e_1} + l_R)w_{x_k}) :\\ &\qquad i \in I \wedge (l_R, r_R) \in R_{e_1}(i) \wedge (s,l,r) \in T_{x_k} \wedge iw_{x_k} \le l \wedge r < (i+1)w_{x_k}\}\end{aligned}$$

where $R_e(i) = Q_{\mathsf{roots}}(\{(s, l - iw_e, r - iw_e) : (s,l,r) \in T_e \wedge iw_e \le l \wedge r < (i+1)w_e\})$. On this representation we define T'_{e_2} inductively. The result is then $T_{\mathsf{for}\, x\in e_1 \,\mathsf{do}\, e_2} = T'_{e_2}$ where $w_{\mathsf{for}\, x\in e_1 \,\mathsf{do}\, e_2} = w_{e_1} \cdot w_{e_2}$. Note that $T_{\mathsf{for}\, x\in e_1 \,\mathsf{do}\, e_2}$ differs from T'_{e_2} only in its *width*—this corresponds to the concatenation of the results of the individual iterations of the $\mathsf{for}\, x \in e_1 \,\mathsf{do}\, e_2$ iterator. Figure 4 summarizes the four steps—the translation (1) of the subexpression e_1, the extension (2) of the original environment, the translation (3) of e_2 with respect to this new environment, and the return (4) to the original environment—involved in the translation of a single $\mathsf{for}\, x \in e_1 \,\mathsf{do}\, e_2$ expression.

[4] In general, we may need several *primed* environment representations during the translation of a FLWR expression that are distinct (i.e., they represent environments for different subexpressions). Technically, we would need to add another level of subscripts to indicate this fact (omitted to improve readability).

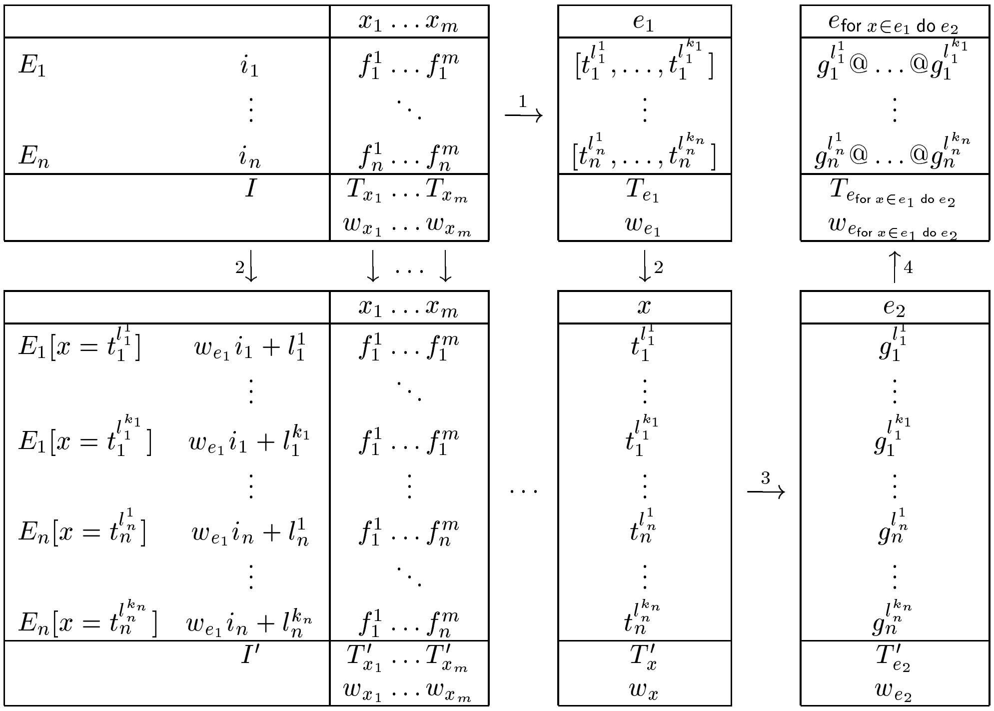

Figure 4: TRANSLATION OF "$\mathsf{for}\, x \in e_1\, \mathsf{do}\, e_2$".

Theorem 4.5 *Let $I, T_{x_1}, \dots, T_{x_m}$ be a representation of the environments $[E_1, \dots, E_n]$ as above and e a FLWR expression over $x_1, \dots, x_m$. Then I, T_e represents $[[\![e]\!] E_1, \dots, [\![e]\!] E_n]$. In addition T_e is a $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$ query (and even a $\mathsf{FO}_{\mathbb{N}}$ query if all builtin operations in e have $\mathsf{FO}_{\mathbb{N}}$ translations).*

5 Implementation Issues

The first issue that needs to be addressed in any practical implementation of the above translation relates to the size of the values used to represent the left and right endpoints in the interval encoding. The following theorem addresses this issue:

Theorem 5.1 *Let e be a FLWR expression over $x_1, \dots, x_m$. Then $w_e \leq f(w_{x_1}, \dots, w_{x_m})$ for f a polynomial function of degree bounded by $|e|$.*

This in particular means that the number of *bits* needed to represent the values l and r throughout the query evaluation is *linear* in the size of the input query (and thus, if the implementation of Nat does not provide sufficient precision, we can allocate additional fixed number of attributes).

There are also several immediate paths to enhance the translation proposed in Section 4 that are specific to handling XML relations and that are thus beyond the capabilities of standard relational optimization and query evaluation techniques:

Schema, Types, etc. The first, and probably most promising, approach relates to employing additional information about the structure of the input XML documents. Such information can be captured by *integrity constraints* (e.g., as present in a DTD or XSchema specification). There are two XML-specific uses of this information:

- we can exploit regularities in the XML documents to create relational representation of the information on higher conceptual level than the level of XML elements. Here we can use the *materialized view* technology (already present in existing relational products [13]) to maintain such conceptual views of an XML document. This observation also provides a bridge between the generic structure-oriented (all documents independent of their structure can be represented) and the constraint-oriented (only documents complying with, e.g., a DTD can be represented) approaches [10] to relational representation of XML: the later can be considered in terms of (possibly materialized) views over the former representation.

- we can utilize *uniqueness* constraints to simplify the translations of several builtin operations. Of particular importance is the simplification of *structural equality* tests: in practice (as witnessed by virtually all the examples in various W3C documents) we only ever use the $\mathsf{FO}_{\mathbb{N}}$ definable *label equality*. While this replacement leads to superior performance, inferring the equivalence between structural and label equalities depends on our ability to detect unique identifiers in XML documents (keys).

Extensions to the Encoding. Another way to combat the need non-first order constructs in the translation is via extending the relational encoding of XML: indeed the use of the interval encoding already successfully evades problems caused by ancestorship queries in trees. Here, of course, we need to balance the increase in our ability to define XML queries by first order queries with the *space* needed for the extended encoding and the obligation to *maintain* the extended encoding in results of operations on XML.

Efficient Relational Operations for Order. Last but not least are the improvements in the actual underlying relational engines. Here, the main improvements come from access methods and operators that can take advantage from ordering of values (numbers). This fact has been already identified in the study of range queries and improvements to the merge-sort join algorithm [14]. The results of that study can be extended to more complex operator patterns resulting from the translation of XML queries. These improvements are closely tied with the ability to infer path order constraints [11] in XML data.

6 Conclusion

The following table summarizes the results achieved by our translation of XML queries from the core fragment of XQuery to queries definable over an interval based encoding of XML documents:

Feature	XQuery	$\mathsf{FO}_{\mathbb{N}}$	$\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$
XML constructors	yes	yes	yes
atomic equality	yes	yes	yes
structural equality	yes	no	yes
internal sorting	yes	partially	yes
XPATH expressions	yes	yes	yes
depth-first traversal	yes	yes	yes
breadth-first traversal	no	no	yes
FLWR(atomic equality)	yes	yes	yes
FLWR(structural equality)	yes	no	yes
document reversal	no	yes	yes
star-free extended regular expressions	no	yes	yes
regular expressions	no	no	yes(conjecture)

The proposed translation extends the applicability of relational technology to querying XML. In particular it shows how to implement transformations previously believed impossible in the relational model and that were often explicitly disallowed or delegated to external processing tools (cf. the notions of *untranslatable expressions* and *tagging templates* in [9]).

There are several directions of future research that derive from our (necessarily limited) discussions in previous sections on implementation issues. In one direction, further evaluation of relational formulations of XQuery queries (on XML relations) can lead to improved relational technology. And in the other direction, existing algorithms and theory for *formatted databases* relating to constraint theory, to query optimization techniques, to materialized views, and so on, can lead to improved XQuery query evaluation and optimization.

References

[1] Michael Benedikt and Leonid Libkin. Relational queries over interpreted structures. *Journal of the ACM*, 47(4):644–680, 2000.

[2] Joe Celko. Trees, Databases and SQL. *DBMS*, 7(10):48–57, 1994. (later varsion in [3]).

[3] Joe Celko. *SQL for Smarties*. Morgan-Kaufmann, 1999 (2th ed.).

[4] Don Chamberlin, James Clark, Daniela Florescu, Jonathan Robie, Jirtme Simèon, and Mugur Stefanescu. XQuery 1.0: An XML Query Language. Technical report, W3C, 2001.

[5] Donald Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. In *The World Wide Web and Databases, 3rd International Workshop WebDB 2000*, pages 1–25. Springer LNCS 1997, 2000.

[6] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. Technical report, W3C, 1999.

[7] Mary Fernandez and Jonathan Marsh. XQuery 1.0 and XPath 2.0 Data Model. Technical report, W3C, 2001.

[8] Thorsten Fiebig and Guido Moerkotte. Evaluating queries on structure with extended access support relations. In D. Suciu and G. Vossen, editors, *Proceedings WebDB 2000*, volume 1997 of *Lecture Notes in Computer Science*, pages 125–136. Springer-Verlag, 2000.

[9] Ioana Manolescu, Daniela Florescu, and Donald Kossmann. Answering xml queries over heterogenous data sources. In Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass, editors, *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 241–250, 2001.

[10] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *The VLDB Journal*, pages 302–314, 1999.

[11] David Toman and Grant E. Weddell. On Attributes, Roles, and Dependencies in Description Logics and the Ackermann Case of the Decision Problem. In *Proceedings of Description Logics, CEUR-WS, vol.49*, pages 76–85, 2001.

[12] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, and Shunsuke Uemura. Xrel: A path-based approach to storage and retrieval of xml documents using relational databases. *ACM Transactions on Internet Technology*, 1(1):110–141, 2001.

[13] Markos Zaharioudakis, Roberta Cochrane, George Lapis, Hamid Pirahesh, and Monica Urata. Answering complex sql queries using automatic summary tables. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 105–116, 2000.

[14] Chun Zhang, Jeffrey Naughton, David DeWitt, Qiong Luo, and Guy Lohman. On supporting containment queries in relational database management systems. In Timos Sellis and Sharad Mehrotra, editors, *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 425–436, 2001.

A Translations of Builtin Functions

First-order definable functions.

$$\begin{array}{rcl}
Q_{[]} &=& \emptyset \\
Q_{\mathsf{XNode}}(X) &=& \{(s,0,w_x+1)\} \cup \{(s,l+1,r+1) : (s,l,r) \in X\} \\
Q_{@}(X_1,X_2) &=& X_1 \cup \{(s,l+w_{x_1},r+w_{x_1}) : (s,l,r) \in X_2\} \\
Q_{\mathsf{head}}(X) &=& \{(s,l,r) : (s,l,r) \in X \wedge (s',l',r') \in X \wedge l' \leq l \wedge r \leq r' \wedge \\
&& \qquad \neg\exists (s'',l'',r'') \in X \wedge l'' < l'\} \\
Q_{\mathsf{tail}}(X) &=& \{(s,l,r) : (s,l,r) \in X \wedge (s',l',r') \in X \wedge l' > l \wedge \\
&& \qquad \neg\exists (s'',l'',r'') \in X \wedge l'' < l'\} \\
Q_{\mathsf{reverse}}(X) &=& \{(s,l-l'+(w_x-r'),r-l'+(w_x-r')) : (s,l,r) \in X \wedge \\
&& \qquad (s',l',r') \in Q_{\mathsf{roots}}(X) \wedge l' \leq l \wedge r \leq r'\} \\
Q_{\mathsf{select}_p}(X) &=& \{(s,l,r) : (s,l,r) \in X \wedge (s',l',r') \in Q_{\mathsf{roots}}(X) \wedge l' \leq l \wedge r \leq r' \wedge s' = p\} \\
Q_{\mathsf{distinct}}(X) &=& \{(s,l,r) : (s,l,r) \in X \wedge (s',l',r') \in Q_{\mathsf{roots}}(X) \wedge l' \leq l \wedge r \leq r' \wedge \\
&& \neg\exists (s'',l'',r'') \in Q_{\mathsf{roots}}(X). r'' < l' \wedge \\
&& \quad Q_{=}(\{(s,l,r) \in X : l' \leq l \wedge r \leq r'\}, \{(s,l,r) \in X : l'' \leq l \wedge r \leq r''\})\} \\
Q_{\mathsf{roots}}(X) &=& \{(s,l,r) : (s,l,r) \in X \wedge \neg\exists s',l',r'.(s',l',r') \in X \wedge l' < l \wedge r < r'\} \\
Q_{\mathsf{children}}(X) &=& \{(s,l,r') : (s,l,r) \in X \wedge \exists s',l',r'.(s',l',r') \in X \wedge l' < l \wedge r < r'\} \\
Q_{\mathsf{subtreesdfs}}(X) &=& \{(s,l+w_x l',r+w_x l') : (s,l,r) \in X \wedge (s',l',r') \in X \wedge l' \leq l \wedge r \leq r'\} \\
Q_{\mathsf{empty}}(X) &=& (\neg\exists s,l,r.(s,l,r) \in X)
\end{array}$$

where $w_{[]} = 0$, $w_{\mathsf{XNode}} = w_x + 2$, $w_{@} = w_{x_1} + w_{x_2}$, $w_{\mathsf{head}} = w_x$, $w_{\mathsf{tail}} = w_x$, $w_{\mathsf{reverse}} = w_x$, $w_{\mathsf{select}_p} = w_x$, $w_{\mathsf{distinct}} = w_x$, $w_{\mathsf{roots}} = w_x$, $w_{\mathsf{children}} = w_x$, and $w_{\mathsf{subtrees}} = w_x^2$.

Structural Equality. There are two essential features that cannot be handled without the use of counting. The first problem we face is the translation of *structural equality* on XML forests. However, results on the expressive power of first-order languages with arithmetic [1] imply the following.

Proposition A.1 *Let R and S be representations of XML forests x and y, respectively. There is no (closed) first-order query Q (with safe arithmetic) such that $\{R,S\} \models Q \iff x = y$.*

We therefore consider non first-order features of SQL, in particular, to the counting aggregate.

Proposition A.2 *Let X be a representation of an XML forest x. Then the $\mathsf{FO}_{\mathbb{N}} + \mathsf{count}$ query*

$$\begin{array}{rl}
\mathsf{XNorm}(X) = \{(s, & |\{(s',l',r') \in X : l' < l\}| + |\{(s',l',r') \in X : r' < l\}|, \\
& |\{(s',l',r') \in X : r' > r\}| + |\{(s',l',r') \in X : l' > r\}|) : X(s,l,r)\}
\end{array}$$

also represents x. Moreover, the representation $\mathsf{XNorm}(X)$ is tight and pointed.

Sorting. The other problematic feature in XML query languages relates to *sorting*, in particular to the use of a sort operator *in a subquery*[5]. However, an interval encoding of trees allows direct access to the interval values—and therefore allows a virtual reordering of XML forests *without the actual use of a sort operator*. To accomplish this goal we need an auxiliary boolean query that implements ordering on representations of XML forests: a formula $Q_<(X_1,X_2)$ that is true if the XML document represented by X_1 is *smaller* than the one represented by X_2.

$$\begin{array}{rcl}
Q_{\mathsf{deepequal}}(X_1,X_2) &=& (\forall s,l,r.(s,l,r) \in \mathsf{XNorm}(X_1) \iff (s,l,r) \in \mathsf{XNorm}(X_2)) \\
Q_{\mathsf{sort}(<)}(X) &=& \{(s,cw_x+l,cw+x+r) : (s,l,r) \in X \wedge \\
&& \quad (s',l',r') \in Q_{\mathsf{roots}}(X) \wedge l' \leq l \wedge r \leq r' \wedge \\
&& \quad c = |\{(s'',l'',r'') \in Q_{\mathsf{roots}}(X) : Q_<(\{(s,l,r) \in X : l'' \leq l \wedge r \leq r''\}, \\
&& \qquad\qquad \{(s,l,r) \in X : l' \leq l \wedge r \leq r'\})|\}
\end{array}$$

[5]Standard SQL allows the sort operator only at the top-level of the query.