# Integrating XQuery and Logic Programming*

Jesús M. Almendros-Jiménez, Antonio Becerra-Terón
and Francisco J. Enciso-Baños

Dpto. Lenguajes y Computación.
Universidad de Almería. {jalmen,abecerra,fjenciso}@ual.es

**Abstract.** In this paper we investigate how to integrate the XQuery language and logic programming. With this aim, we represent XML documents by means of a logic program. This logic program represents the document schema by means of rules and the document itself by means of facts. Now, XQuery expressions can be integrated into logic programming by considering a translation from for-let-where-return expressions into logic rules and a goal.

## 1 Introduction

*XQuery* [W3C07b,CDF+04,Wad02,Cha02] is a typed functional language devoted to express queries against XML documents. It contains *XPath 2.0* [W3C07a] as a sublanguage. *XPath 2.0* supports navigation, selection and extraction of fragments from XML documents. *XQuery* also includes expressions to construct new XML values and to join multiple documents. The design of *XQuery* has been influenced by group members with expertise in the design and implementation of other high-level languages. *XQuery* has static typed semantics and a formal semantics which is part of the *W3C* standard [CDF+04,W3C07b].

The integration of *declarative programming* and *XML data processing* is a research field of increasing interest in the last years (see [BBFS05] for a survey). There are proposals of new languages for XML data processing based on functional, and logic programming. In addition, *XPath* and *XQuery* have been also implemented in declarative languages.

The most relevant contribution is the *Galax* project [MS03,CDF+04], which is an implementation of *XQuery* in functional programming, using *OCAML* as host language. There are also proposals for new languages based on functional programming rather than implementing *XQuery*. This is the case of *XDuce* [HP03] and *CDuce* [BCF05], which are languages for XML data processing, using regular expression pattern matching over XML trees, subtyping as basic mechanism, and *OCAML* as host language. The *CDuce* language does fully statically-typed transformation of XML documents, thus guaranteeing correctness. In addition, there are proposals around *Haskell* for the handling of XML documents, such as *HaXML* [Thi02] and [WR99].

* This work has been partially supported by the EU (FEDER) and the Spanish MEC under grant TIN2005-09207-C03-02.

There are also contributions in the field of use logic programming for the handling of XML documents. For instance, the *Xcerpt project* [SB02] proposes a pattern and rule-based query language for XML documents, using the so-called query terms including logic variables for the retrieval of XML elements. For this new language a specialized unification algorithm for query terms has been studied. Another contribution of a new language is *XPathLog* (the *LOPIX* system) [May04] which is a *Datalog*-style extension of *XPath* with variable bindings. This is also the case of *XCentric* [CF03], which can represent XML documents by means of logic programming, and handles XML documents by considering terms with functions of flexible arity and regular types. Finally, *FNPath* [Sei02] is a proposal for using Prolog as query language for XML documents based on a field-notation, for evaluating *XPath* expressions based on *DOM*. The Rule Markup Language (*RULEML*) [Bol01,Bol00] is a different kind of proposal in the research area. The aim of the approach is the representation of Prolog facts and rules into XML documents, and thus, the introduction of *rule systems* into the *Web*. Finally, some well-known Prolog implementations include libraries for loading and querying XML documents, such as *SWI-Prolog* [Wie05] and *CIAO* [CH01].

In this paper, we investigate how to integrate the *XQuery* language and logic programming. With this aim:

1. A XML document can be seen as a logic program, by considering *facts* and *rules* for expressing both the XML schema and document. This approach was already studied in our previous work [ABE07,ABE06].
2. A *XQuery expression can be translated into logic programming* by considering a set of rules and a specific goal. Taking as starting point the translation of *XPath* of our previous work [ABE07,ABE06], the translation of *XQuery* introduces *new rules* for the *join* of documents, and for the translation of *for-let-where-return* expressions into logic programming. In addition, a *specific goal* is generated for obtaining the answer to an *XQuery* query.
3. Our technique allows the handling of XML documents as follows. Firstly, the XML documents are loaded. It involves the translation of the XML documents into a logic program. For efficiency reasons the rules, which correspond to the XML document structure, are loaded in *main memory*, but facts, which represent the values of the XML document, are stored in *secondary memory*, whenever they do not fit in main memory (using appropriate *indexing techniques*). Secondly, the user can now write queries against the loaded documents. Now, *XQuery* queries are translated into a logic program and a specific goal. The evaluation of such goal uses the indexing in order to improve the efficiency of query solving. Finally, the answer of the goal can be represented by means of an output XML document.

As far as we know, this is the first time that *XQuery* is implemented in logic programming. Previous proposals either define new query languages for XML documents in logic and functional programming or implement *XQuery*, but in functional programming. The advantages of such proposal is that *XQuery* is embedded into logic programming, and thus *XQuery* can be combined with logic programs. For instance, logic programming can be used as inference engine, one of the requirements of the so-called *Semantic Web* (`http://www.w3.org/2001/sw/`), in the line of *RuleML*.

Our proposal requires the representation of XML documents into logic programming, which can be compared with those ones representing XML documents in logic programming (for instance, [SB02,CF03]) and, with those ones representing XML documents in relational databases (for instance, [BGvK+05]). In our case, rules are used for expressing the structure of well-formed XML documents, and XML elements are represented by means of facts. Moreover, our handling of XML documents is more "database-oriented" since we use secondary memory and file indexing for selective reading of records. The reason for such decision is that XML documents can usually be too big for main memory [MS03]. Our proposal uses as basis the implementation of *XPath* in logic programming studied in our previous work [ABE07]. In addition, we have studied how to consider a bottom-up approach to the same proposal of [ABE07] in [ABE06].

The structure of the paper is as follows. Section 2 will present the translation of XML documents into Prolog; section 3 will review the translation of *XPath* into logic programming; section 4 will provide the new translation of *XQuery* expressions into logic programming; and finally, section 5 will conclude and present future work. A more complete version of our paper (with a bigger set of examples) can be found in `http://www.ual.es/~jalmen`.

## 2 Translating XML Documents into Logic Programming

In order to define our translation we need to number the nodes of the XML document. A similar numbering has been already adopted in some proposals for representing XML in relational databases [OOP+04,TVB+02,BGvK+05].

Given an XML document, we can consider a new XML document called *node-numbered XML document* as follows. Starting from the root element numbered as 1, the node-numbered XML document is numbered using an attribute called **nodenumber**[1] where each $j$-th child of a tagged element is numbered with the sequence of natural numbers $i_1.\ldots.i_t.j$ whenever the parent is numbered as $i_1.\ldots.i_t$: $< tag\ att_1 = v_1, \ldots, att_n = v_n, \mathbf{nodenumber} = \mathbf{i_1}.\ldots.\mathbf{i_t}.\mathbf{j} >$ $elem_1, \ldots, elem_s < /tag >$. This is the case of tagged elements. If the $j$-th child is of a basic type (non tagged) and the parent is an inner node, then the element is labeled and numbered as follows: $< unlabeled\ \mathbf{nodenumber} = \mathbf{i_1}.\ldots.\mathbf{i_t}.\mathbf{j} >$ $elem < /unlabeled >$; otherwise the element is not numbered. It gives to us a *hierarchical and left-to-right numbering* of the nodes of an XML document. An element in an XML document is further left in the XML tree than another when the node number is smaller w.r.t. the lexicographic order of sequence of natural numbers. Any numbering that identifies each inner node and leaf could be adapted to our translation.

In addition, we have to consider a new document called *type and node-numbered XML document* numbered using an attribute called **typenumber** as

[1] It is supposed that "nodenumber" is not already used as attribute in the tags of the original XML document.

follows. Starting the numbering from 1 in the root of the node-numbered XML document, each tagged element is numbered as: $< tag\ att_1 = v_1, \ldots, att_n = v_n, nodenumber = i_1. \ldots, i_t.j, \mathbf{typenumber} = \mathbf{k} > elem_1, \ldots, elem_s < /tag >$. The type number $k$ of the tag is equal to $l + n + 1$ whenever the type number of the parent is $l$, and $n$ is the number of tagged elements weakly distinct [2] occurring in leftmost positions at the same level of the XML tree [3].

Now, the translation of the XML document into a logic program is as follows. For each inner node in the type and node numbered XML document $< tag\ att_1 = v_1, \ldots, att_n = v_n, nodenumber = i, typenumber = k > elem_1, \ldots, elem_s < /tag >$ we consider the following rule, called *schema rule*:

**Schema Rule in Logic Programming**

$$
\begin{aligned}
& tag(tagtype(Tag_{i_1}, \ldots, Tag_{i_t}, [Att_1, \ldots, Att_n]), NTag, k, Doc) \text{:-} \\
& \qquad tag_{i_1}(Tag_{i_1}, [NTag_{i_1}|NTag], r, Doc), \\
& \qquad \ldots, \\
& \qquad tag_{i_t}(Tag_{i_t}, [NTag_{i_t}|NTag], r, Doc), \\
& \qquad att_1(Att_1, NTag, r, Doc), \\
& \qquad \ldots, \\
& \qquad att_n(Att_n, NTag, r, Doc).
\end{aligned}
$$

where *tagtype* is a new function symbol used for building a Prolog term containing the XML document; $\{tag_{i_1}, \ldots, tag_{i_t}\}$, $i_j \in \{1, \ldots, s\}$, $1 \leq j \leq t$, is the *set of tags* of the tagged elements $elem_1, \ldots, elem_s$; $Tag_{i_1}, \ldots, Tag_{i_t}$ are variables; $att_1, \ldots, att_n$ are the attribute names; $Att_1, \ldots, Att_n$ are variables, one for each attribute name; $NTag_{i_1}, \ldots, NTag_{i_t}$ are variables (used for representing the last number of the node number of the children); $NTag$ is a variable (used for representing the node number of *tag*); $k$ is the type number of *tag*; and finally, $r$ is the type number of the tagged elements $elem_1, \ldots, elem_s$ [4].

In addition, we consider facts of the form: $att_j(v_j, i, k, doc)$ $(1 \leq j \leq n)$, where *doc* is the name of the document. Finally, for each leaf in the type and node numbered XML document: $< tag\ nodenumber = i, typenumber = k > value < /tag >$, we consider the *fact*: $tag(value, i, k, doc)$. For instance, let us consider the following XML document called "books.xml":

**XML document**

```
<books>
   <book year="2003">
      <author>Abiteboul</author>
      <author>Buneman</author>
      <author>Suciu</author>
      <title>Data on the Web</title>
      <review>A <em>fine</em> book.</review>
   </book>
```

[2] Two elements are weakly distinct whenever they have the same tag but not the same structure.

[3] In other words, type numbering is done by levels and in left-to-right order, but each occurrence of weakly distinct elements increases the numbering in one unit.

[4] Let us remark that since *tag* is a tagged element, then $elem_1, \ldots, elem_s$ have been tagged with "unlabeled" labels in the type and node numbered XML document when they were not labeled; thus they must have a type number.

```
    <book year="2002">
        <author>Buneman</author>
        <title>XML in Scotland</title>
        <review><em>The <em>best</em> ever!</em></review>
    </book>
</books>
```

Now, the previous XML document can be represented by means of a logic program as follows:

**Translation into Prolog of an XML document**

| Rules (Schema): | Facts (Document): |
|---|---|
| books(bookstype(Book, []), NBooks,1,Doc) :- book(Book, [NBook\|NBooks],2,Doc). | year('2003', [1, 1], 3,"books.xml"). |
| book(booktype(Author, Title, Review, [Year]), NBook ,2,Doc) :- author(Author, [NAu\|NBook],3,Doc), title(Title, [NTitle\|NBook],3,Doc), review(Review, [NRe\|NBook],3,Doc), year(Year, NBook,3,Doc). | author('Abiteboul', [1, 1, 1], 3,"books.xml"). author('Buneman', [2,1, 1], 3,"books.xml"). author('Suciu', [3,1,1], 3,"books.xml"). title('Data on the Web', [4, 1, 1], 3,"books.xml"). unlabeled('A', [1, 5, 1, 1], 4,"books.xml"). em('fine', [2, 5, 1, 1], 4,"books.xml"). |
| review(reviewtype(Un,Em,[]),NReview,3,Doc):- unlabeled(Un,[NUn\|NReview],4,Doc), em(Em,[NEm\|NReview],4,Doc). | unlabeled('book.', [3, 5, 1, 1], 4,"books.xml"). year('2002', [2, 1], 3,"books.xml"). author('Buneman', [1, 2, 1], 3,"books.xml"). |
| review(reviewtype(Em,[]),NReview,3,Doc):- em(Em,[NEm\|NReview],5,Doc). | title('XML in Scotland', [2, 2, 1], 3,"books.xml"). unlabeled('The', [1, 1, 3, 2, 1], 6,"books.xml"). |
| em(emtype(Unlabeled,Em,[]),NEms,5,Doc) :- unlabeled(Unlabeled,[NUn\|NEms],6,Doc), em(Em, [NEm\|NEms],6,Doc). | em('best', [2, 1, 3, 2, 1], 6,"books.xml"). unlabeled('ever!', [3, 1, 3, 2, 1], 6,"books.xml"). |

Here we can see the translation of each tag into a predicate name: *books*, *book*, etc. Each predicate has four arguments, the first one, used for representing the XML document structure, is encapsulated into a function symbol with the same name as the tag adding the suffix *type*. Therefore, we have *bookstype*, *booktype*, etc. The second argument is used for numbering each node; the third argument of the predicates is used for numbering each type; and the last argument represents the document name. The key element of our translation is to be able to recover the original XML document from the set of rules and facts.

## 3 Translating XPath into Logic Programming

In this section, we present how *XPath* expressions can be translated into a logic program. Here we present the basic ideas, a more detailed description can be found in [ABE07].

We restrict ourselves to *XPath* expressions of the form $xpathexpr = /expr_1 \ldots /expr_n$ where each $expr_i$ can be a tag or a *boolean condition* of the form $[xpathexpr = value]$, where $value$ has a basic type. More complex *XPath* queries [W3C07a] will be expressed in *XQuery*, and therefore they will be translated in next section.

With the previous assumption, each *XPath* expression $xpathexpr = /expr_1 \ldots /expr_n$ defines a *free of equalities XPath expression*, denoted by $FE(xpathexpr)$. This free of equalities *XPath* expression defines a subtree of the XML document, in which is required that some paths exist (occurences of boolean conditions $[xpathexpr]$). For instance, with respect to the *XPath* expression */books/book*

$[author = Suciu]/title$, the free of equalities *XPath* expression is $/books/book$ $[author]$ $/title$ and the subtree of the type and node numbered XML document which corresponds with the expression $/books/book$ $[author]/title$ is as follows:

**Subtree defined by a Free of Equalities XPath Expression**

*<books nodenumber=1, typenumber=1>*
*<book year="2003", nodenumber=1.1, typenumber=2>*
*<author nodenumber=1.1.1 typenumber=3>Abiteboul</author>*
*<author nodenumber=1.1.2 typenumber=3>Buneman</author>*
*<author nodenumber=1.1.3 typenumber=3>Suciu</author>*
*<title nodenumber=1.1.4 typenumber=3>Data on the Web</title>*
*</book>*
*<book year="2002" nodenumber=1.2, typenumber=2>*
*<author nodenumber=1.2.1 typenumber=3>Buneman</author>*
*<title nodenumber=1.2.2 typenumber=3>XML in Scotland</title>*
*</book>*
*</books>*

Now, given a type and node numbered XML document $\mathcal{D}$, a program $\mathcal{P}$ representing $\mathcal{D}$, and an *XPath* expression *xpathexpr* then the *logic program obtained from xpathexpr* is $\mathcal{P}^{xpathexpr}$, obtained from $\mathcal{P}$ taking the schema rules and facts for the subtree of $\mathcal{D}$ defined by $FE(xpathexpr)$. For instance, with respect to the above example, the schema rules defined by $/books/book$ $[author]/title$ are:

**Translation into Prolog of an XPath Expression**

```
books(bookstype(Book, []), NBooks, 1,Doc):-
        book(Book, [NBook|NBooks], 2,Doc).
book(booktype(Author,Title,Review,[Year]),NBook,2,Doc) :-
        author(Author,[NAuthor|NBook],3,Doc),
        title(Title,[NTitle|NBook],3,Doc).
```

and the facts, the set of facts for *title* and *author*. Let us remark that in practice, these rules can be obtained from the schema rules by removing predicates, that is, removing the predicates in the schema rules which are not tags in the free of equalities *XPath* expression.

Now, given a type and node numbered XML document, and an *XPath* expression *xpathexpr*, the *goals obtained from xpathexpr* are defined as follows. Firstly, each *XPath* expression *xpathexpr* can be mapped into a set of prolog terms, denoted by $PT(xpathexpr)$, representing the *pattern* of the query [5]. Basically, the pattern represents the required structure of the record. Now, the *goals* are defined as: $\{: -tag(Tag, Node, r, doc)\ \{Tag \rightarrow t\ \} |\ t \in PT(xpathexpr),$ $r\ is\ a\ type\ number\ of\ tag\ for\ t\}$ where *tag* is the leftmost tag in *xpathexpr* with a boolean condition (and it is the rightmost tag whenever boolean conditions do not exist); $Tag$ and $Node$ are variables; and *doc* is the document name.

For instance, with respect to $/books/book\ [author = Suciu]/title$, $PT(/books/$ $book\ [author = Suciu]/title) = \{booktype('Suciu', Title, Review, [Year])\}$, and therefore the (unique) goal is $: -book(booktype('Suciu', Title, Review, Year), Node,$ $2, "books.xml")$.

We will call to the leftmost tag with a boolean condition the *head tag* of *xpathexpr* and is denoted by $htag(xpathexpr)$. In the previous example, $htag(/books/book[author = Suciu]/title) = book$.

[5] Due to XML records can have different structure, one pattern is generated for each kind of record.

In summary, the handling of an *XPath* query involves the *"specialization"* of the schema rules of the XML document and the *generation* of one or more goals. The goals are obtained from the leftmost tag with a boolean condition on the *XPath* expression. Obviously, instead of a set of goals for each *XPath* expression, a unique goal can be considered by adding new rules. In the following we will assume this case.

## 4 Translating XQuery into Logic Programming

Similarly to *XPath*, an *XQuery* expression is translated into a logic program and generates a specific goal. We focus on the *XQuery* core language, whose grammar can be defined as follows.

**Core XQuery**

*xquery*:= $dxpfree| < tag >' \{'xquery, \ldots, xquery'\}' < /tag > |flwr.$
*dxpfree*:= *document(doc)* '/' *xpfree.*
*flwr*:= **for** *\$var* **in** *vxpfree [***where** *constraint]* **return** *xqvar*
| **let** *\$var* **:=** *vxpfree [***where** *constraint]* **return** *xqvar.*
*xqvar*:= $vxpfree| < tag >' \{'xqvar, \ldots, xqvar'\}' < /tag > |flwr.$
*vxpfree*:= *\$var* | *\$var* '/' *xpfree* | *dxpfree.*
*Op*:= <= | >= | < | > | =.
*constraint* := *vxpfree Op value* | *vxpfree Op vxpfree*
| *constraint* '**or**' *constraint* | *constraint* '**and**' *constraint.*

where $value$ is an XML document, *doc* is a document name, and $xpfree$ is a free of equalities *XPath* expression. Let us remark that *XQuery* expressions use free of equalities *XPath* expressions, given that equalities can be always introduced in *where* expressions. Finally, we will say that an *XQuery* expression *ends with attribute name* in the case of the *XQuery* expression has the form *vxpfree* and the rightmost element has the form *@att*, where *att* is an attribute name. The translation of an *XQuery* expression consists of three elements.

- Firstly, for each *XQuery* expression $xquery$, we can define analogously to *XPath* expressions, the so-called *head tag*, denoted by $htag(xquery)$, which is the *predicate name* used for the building of the goal (or subgoal whenever the expression *xquery* is nested).
- Secondly, for each *XQuery* expression $xquery$, we can define the so-called *tag position*, denoted by $tagpos(xquery)$, representing the argument of the head tag (i.e. the argument of the predicate name) in which the answer is retrieved.
- Finally, for each *XQuery* expression $xquery$ we can define a logic program $\mathcal{P}^{xquery}$ and a specific goal.

In other words, each *XQuery* expression can be mapped in the translation into a program $\mathcal{P}^{xquery}$ and into a goal of the form $: -tag(Tag_1, \ldots, Tag_n, Node, Type, Docs)$ where $tag$ is the head tag, and $Tag_{pos}$ represents the answer of the query, where $pos = tagpos(xquery)$. In addition, $Node$ and $Type$ represent the node and type numbering of the output document, and $Docs$ represents the documents involved in the query. The above elements are defined in Tables 2 and 3 for each case, assuming the notation of Table 1.

**Table 1.** Notation

| |
|---|
| $Vars(\Gamma) = \{\$var \mid (\$var, let, vxpfree, C) \in \Gamma \ or \ (\$var, for, vxpfree, C) \in \Gamma\}$; |
| $Doc(\$var, \Gamma) = doc$ whenever $\bar{\Gamma}_{\$var} = document(doc)/xpfree$; |
| $DocVars(\Gamma) = \{\$var \mid (\$var, let, dxpfree, C) \in \Gamma \ or \ (\$var, for, dxpfree, C) \in \Gamma\}$; |
| $\Gamma_{\$var} = vxpfree$ whenever $(\$var, let, vxpfree, C)$ or $(\$var, for, vxpfree, C) \in \Gamma$; |
| $\bar{\Gamma}_{\$var} = vxpfree[\lambda_1 \cdot \ldots \cdot \lambda_n]$ where $\lambda_i = \{\$var_i \to \Gamma_{\$var_i}\}$ and $\{\$var_1, \ldots, \$var_n\} = Vars(\Gamma)$; |
| $Root(\$var) = \$var'$ whenever $\$var \in DocVars(\Gamma)$ and $\$var = \$var'$ or $(\$var, let, \$var''/xpfree, C) \in \Gamma \ or \ (\$var, for, \$var''/xpfree, C) \in \Gamma$ and $Root(\$var'') = \$var'$; |
| $Rootedby(\$var, \mathcal{X}) = \{xpfree \mid \$var/xpfree \in \mathcal{X}\}$; |
| $Rootedby(\$var, \Gamma) = \{xpfree \mid \$var/xpfree \ Op \ \ vxpfree \in C$ or $\$var/xpfree \ \ Op \ value \in C, C \in Constraints(\$var, \Gamma)\}$; and |
| $Constraints(\$var, \Gamma) = \{C_i \mid C \equiv C_1 \ Op \ \ldots \ Op \ C_n$, $(\$var, let, vxpfree, C) \in \Gamma \ or \ (\$var, for, vxpfree, C) \in \Gamma\}$ |

### 4.1 Examples

Let us suppose a query requesting the year and title of the books published before 2003.

```
xquery = for $book in document ('books.xml')/books/book
            return let $year := $book/@year
            where $year<2003
            return <mybook>{$year, $book/title}</mybook>
```

For this query, the translation is as follows:

$$\mathcal{P}^{xquery} = \mathcal{P}^{xquery_2}_{(book, for, document('books.xml')/books/book, \emptyset)} =$$

$$\mathcal{P}^{xquery_3}_{(book, for, document('books.xml')/books/book, \emptyset), (\$year, let, \$book/@year, \$year<2003)} =$$

$$\{\mathcal{R}\} \cup \mathcal{P}^{\$year, \$book/title}_{(book, for, document('books.xml')/books/book, \emptyset), (\$year, let, \$book/@year, \$year<2003)}$$

$$\mathcal{R} =$$

$$\boxed{\begin{array}{l} mybook(mybooktype(Title, [Year]), [Node], [Type], [Doc]) :- \\ \qquad join(Title, Year, Node, Type, Doc). \end{array}}$$

$\%\ \{join\} = \{htag(\$year), htag(\$book/title)\};$
$\%\ tagpos(\$year) = 1 \ and \ tagpos(\$book/title) = 2$

$$\mathcal{P}^{\$year, \$book/title}_{(\$book, for, document('books.xml')/books/book, \emptyset), (\$year, let, \$book/@year, \$year<2003)} =$$

$$\{\mathcal{J}^{\Gamma}\} \cup \mathcal{C}^{\Gamma} \cup \{\mathcal{R}^{\$book}\}$$

$$\cup \mathcal{P}^{document('books.xml')/books/book/@year} \cup \mathcal{P}^{document('books.xml')/books/book/title}$$

$$\mathcal{J}^{\Gamma} =$$

$$\boxed{\begin{array}{l} join(Title, Year, [Node], [Type], [Doc]) :- \\ \qquad vbook(Title, Year, Node, Type, Doc), \\ \qquad constraints(vbook(Title, Year)). \end{array}}$$

$\%\ DocVars(\Gamma) = \{\$book\},\ \$year, \$book/title \in \mathcal{X}$
$\%\ Root(\$year) = \$book,\ Root(\$book) = \$book.$

$$\mathcal{C}^{\Gamma} =$$

$$\boxed{\begin{array}{l} constraints(Vbook) :- lc_1^1(Vbook). \\ lc_1^1(Vbook) :- c_1^1(Vbook). \\ c_1^1(vbook(Title, Year)) :- leq(Year, 2003). \end{array}}$$

$\%\ C^1 \equiv c_1^1,\ c_1^1 \equiv \$year < 2003$
$\%\ C^1 \in constraints(\$year, \Gamma) \ and \ Root(\$year) = \$book$

$$\mathcal{R}^{\$book} =$$

$$\boxed{\begin{array}{l} vbook(Title, Year, [Node, Node], [TTitle, TYear], 'books.xml') :- \\ \qquad title(Title, [NTitle \mid Node], TTitle, 'books.xml'), \\ \qquad year(Year, Node, TYear, 'books.xml'). \end{array}}$$

**Table 2.** Translation of XQuery into Logic Programming

| | |
|---|---|
| $\mathcal{P}^{document(doc)/xpfree} =_{def} \mathcal{P}^{xpfree}$<br>$htag(document(doc)/xpfree) =_{def} htag(xpfree)$<br>$tagpos(document(doc)/xpfree) =_{def} tagpos(xpfree)$ | |
| $\mathcal{P}^{<tag>\{xquery\_1,\ldots,xquery\_n\}</tag>} =_{def}$<br>$\quad\{\mathcal{R}\} \cup_{1\leq i\leq n} \mathcal{P}^{xquery_i}$<br>$\mathcal{R} \equiv$<br>$\quad tag(tagtype(Tag^{1}_{p_1},\ldots,Tag^{k}_{p_k},[Att^{1}_{q_1},\ldots,Att^{s}_{q_s}]),$<br>$\qquad [NTag_1,\ldots,NTag_k,NAtt_1,\ldots,NAtt_s],$<br>$\qquad [TTag_1,\ldots,TTag_k,TAtt_1,\ldots,TAtt_s],$<br>$\qquad [DTag_1,\ldots,DTag_k,DAtt_1,\ldots,DAtt_s]) :-$<br>$\qquad tag_1(\overline{Tag^1},NTag_1,TTag_1,DTag_1),$<br>$\qquad \ldots$<br>$\qquad tag_k(\overline{Tag^k},NTag_k,TTag_k,DTag_k),$<br>$\qquad att_1(\overline{Att^1},NAtt_1,TAtt_1,DAtt_1),$<br>$\qquad \ldots$<br>$\qquad att_s(\overline{Att^s},NAtt_s,TAtt_s,DAtt_s).$<br><br>$htag(xquery) =_{def} tag,\ tagpos(xquery) =_{def} 1$ | $\overline{Tag^t}\ 1\leq t\leq k,$<br>*denotes* $Tag^t_1,\ldots,Tag^t_r$<br>*where* $r$ *is the arity of* $tag_t$;<br>$\overline{Att^j}\ 1\leq j\leq s,$<br>*denotes* $Att^j_1,\ldots,Att^j_s$<br>*where* $s$ *is the arity of* $att_j$;<br>$htag(xquery_j) = att_i,\ 1\leq i\leq s,$<br>*for some* $j\in\{1,\ldots,n\}$<br>*which ends with attribute names,*<br>$htag(xquery_j) = tag_t,\ 1\leq t\leq k,$<br>*otherwise*<br>$tagpos(xquery_j) = q_i$ *and*<br>$tagpos(xquery_j) = p_t$<br>*in the same cases.* |
| $\mathcal{P}^{\textbf{for}\ \$var\ \textbf{in}\ vxpfree\ [\textbf{where}\ C]\ \textbf{return}\ xqvar} =_{def}$<br>$\quad\mathcal{P}^{xqvar}_{\{(\$var,for,vxpfree,C)\}}$<br>$htag\ (xquery) =_{def} htag(xqvar)$<br>$tagpos(xquery) =_{def} tagpos(xqvar)$ | |
| $\mathcal{P}^{\textbf{let}\ \$var\ :=\ vxpfree\ [\textbf{where}\ C]\ \textbf{return}\ xqvar} =_{def}$<br>$\quad\mathcal{P}^{xqvar}_{\{(\$var,let,vxpfree,C)\}}$<br>$htag\ (xquery) =_{def} htag(xqvar)$<br>$tagpos(xquery) =_{def} tagpos(xqvar)$ | |
| $\mathcal{P}^{\mathcal{X}}_{\Gamma} =_{def}$<br>$\quad\{\mathcal{R}\}\cup\ \mathcal{P}_{\Gamma}^{\mathcal{X}-\{xquery/xpfree\}\cup_{1\leq i\leq n}\{xqvar_i/xpfree_0\}}$<br>$\mathcal{R} \equiv$<br>$\quad tag(tagtype(Tag_1,\ldots,Tag_r,[Att^1,\ldots,Att^m]),$<br>$\qquad [Node_1,\ldots,Node_s],$<br>$\qquad [Type_1,\ldots,Type_s],$<br>$\qquad [Doc_1,\ldots,Doc_s]) :-$<br>$\qquad tag_1(\overline{Tag^1},Node_1,Type_1,Doc_1),$<br>$\qquad \ldots$<br>$\qquad tag_s(\overline{Tag^s},Node_s,Types_s,Doc_s).$<br>$htag(xquery/xpfree) =_{def} tag$<br>$tagpos(xquery/xpfree) =_{def} 1$ | $xquery \equiv$<br>$< tag > \{xqvar_1,\ldots,$<br>$xqvar_n\} < /tag >$<br>$xquery/xpfree \in \mathcal{X}$<br>$xpfree \equiv /tag/xpfree_0$<br>$\{tag_1,\ldots,tag_s\} =$<br>$\{htag\ (xqvar_1\ /xpfree_0),$<br>$\ldots,\ htag\ (xqvar_n\ /xpfree_0)\}$;<br>$Tag_i = Tag^j_{p_j},\ 1\leq i\leq r,$ *whenever*<br>$tagpos(xqvar_p\ /xpfree_0) = p_j,$<br>$htag(xqvar_p\ /xpfree_0) = tag_j,$<br>$p\in\{1,\ldots,n\}$;<br>$Att_l = Tag^j_{p_j}\ 1\leq l\leq s,$ *whenever*<br>$tagpos(xqvar_p\ /xpfree_0) = p_j,$<br>$htag(xqvar_p\ /xpfree_0) = tag_j$<br>$p\in\{1,\ldots,n\}$<br>$xqvar_p\ /xpfree_0$<br>*ends with attribute names* |
| $\mathcal{P}^{\mathcal{X}}_{\Gamma} =_{def} \mathcal{P}^{\mathcal{X}-\{xquery/xpfree\}\cup\{xqvar/xpfree\}}_{\Gamma\cup\{(\$var,for,vxpfree,C)\}}$<br>$htag(xquery/xpfree) =_{def} htag(xqvar/xpfree)$<br>$tagpos(xquery/xpfree) =_{def} tagpos(xqvar/xpfree)$ | $xquery \equiv$<br>**for** $\$var$ **in** $vxpfree$ [**where** $C$]<br>**return** $xqvar$<br>$xquery/xpfree \in \mathcal{X}$ |
| $\mathcal{P}^{\mathcal{X}}_{\Gamma} =_{def} \mathcal{P}^{\mathcal{X}-\{xquery/xpfree\}\cup\{xqvar/xpfree\}}_{\Gamma\cup\{(\$var,let,vxpfree,C)\}}$<br>$htag(xquery/xpfree) =_{def} htag(xqvar/xpfree)$<br>$tagpos(xquery/xpfree) =_{def} tagpos(xqvar/xpfree)$ | $xquery \equiv$<br>**let** $\$var := vxpfree$ [**where** $C$]<br>**return** $xqvar$<br>$xquery/xpfree \in \mathcal{X}$ |

$\%\ "books.xml" = Doc(\$book,\Gamma)$
$\%\ htag(document\ ('books.xml')\ /books/book\ /@year) = year$
$\%\ htag(document\ ('books.xml')\ /books/book\ /title) = title$
$\%\ \overline{\Gamma}_{\$year} = document\ ('books.xml')\ /books/book/$
$\%\ \overline{\Gamma}_{\$book} = document\ ('books.xml')\ /books/book/$
$\%\ \$year, \$book/title \in \mathcal{X}$
$\mathcal{P}^{document('books.xml')/books/book/@year} = \emptyset$
$\mathcal{P}^{document('books.xml')/books/book/title} = \emptyset$

**Table 3.** Translation of XQuery into Logic Programming (cont'd)

| | |
|---|---|
| $\mathcal{P}^{\mathcal{X}}_{\Gamma} =_{def} \bigcup_{\substack{\$var \in DocVars(\Gamma), \\ \$var = Root(\$var'), \\ xpfree \in Rootedby(\$var',\mathcal{X}) \cup Rootedby(\$var',\Gamma)}} \{\mathcal{J}^{\Gamma}\} \cup \mathcal{C}^{\Gamma} \cup \{\mathcal{R}^{\$var} \mid \$var \in DocVars(\Gamma)\} \quad \mathcal{P}^{\bar{\Gamma}_{\$var/xpfree}}$ *(1)* | *(1) –* $\mathcal{X}$ *does not includes tagged elements and flwr expressions* |
| $\mathcal{J}^{\Gamma} \equiv$<br>$join(Tag_1,\ldots,Tag_m,[Node_1,\ldots,Node_n],$<br>$[Type_1,\ldots,Type_n],[Doc_1,\ldots,Doc_n]) :-$<br>$vvar_1(\overline{Tag^1}, Node_1, Type_1, Doc_1),$ *(2)*<br>$\ldots$<br>$vvar_n(\overline{Tag^n}, Node_n, Type_n, Doc_n),$<br>$constraints(vvar_1(\overline{Tag^1}),\ldots,vvar_n(\overline{Tag^n})).$ | *(2)*<br>– $\{\$var_1,\ldots,\$var_n\} = DocVars(\Gamma);$<br>– $Tag_j = Tag^i_{p_j}$<br>$\$var'/xpfree_j \in \mathcal{X}$<br>$Root(\$var') = \$var_i$<br>*one* $p_j$<br>*for each* $\$var'/xpfree_j$<br>– $\overline{Tag^i} = Tag^i_1 \ldots Tag^i_s$<br>$Tag^i_r,\ 1 \leq r \leq s$<br>*one* $Tag^i_r$<br>*for each* $\$var'/xpfree_r \in \mathcal{X}$<br>$Root(\$var') = \$var_i$<br>*and*<br>*one* $Tag^i_r$<br>*for each* $\$var'/xpfree_r$ *in* $\Gamma$<br>$Root(\$var') = \$var_i$ |
| $\mathcal{R}^{\$var} \equiv$<br>$vvar(Tag_1,\ldots,Tag_n, Node, [Type_1,\ldots,Type_n], doc) :-$<br>$tag_1(Tag_1,[Node_{11},\ldots,Node_{1k_1} \mid NTag], Type_1, doc),$ *(3)*<br>$\ldots,$<br>$tag_n(Tag_n,[Node_{n1},\ldots,Node_{nk_n} \mid NTag], Type_n, doc).$ | *(3)*<br>– $doc = Doc(\$var,\Gamma)$<br>– $tag_i = htag(\overline{\Gamma}_{\$var}/xpfree)$<br>$\$var'\ /xpfree \in \mathcal{X}$<br>$\$var = Root(\$var')$<br>– $Node = [N_1,\ldots,N_n]$<br>$N_i = [Node_{ik_i} \mid NTag]$<br>*if* $(\$var',\ for, vxpfree, C) \in \Gamma$<br>$N_i = NTag$<br>*otherwise* |
| $\mathcal{C}^{\Gamma} \equiv \{$<br>$constraints(Vvar_1,\ldots,Vvar_n) :-$<br>$lc^1_1(Vvar_1,\ldots,Vvar_n),$<br>$\ldots$ *(4)*<br>$lc^n_1(Vvar_1,\ldots,Vvar_n).$<br>$\} \cup_{\$var \in Vars(\Gamma), C^j \in constraints(\$var,\Gamma)} \mathcal{C}^j$ | *(4)* $\{\$var_1,\ldots,\$var_n\} = DocVars(\Gamma);$ |
| $\mathcal{C}^j \equiv$<br>$\{lc^j_i(Vvar_1,\ldots,Vvar_n) :-$<br>$c^j_i(Vvar_1,\ldots,Vvar_n), lc^j_{i+1}(Vvar_1,\ldots,Vvar_n).$<br>$\mid 1 \leq i \leq n, Op_i = \mathbf{and}\}$<br>$\cup$ *(5)*<br>$\{lc^j_i(Vvar_1,\ldots,Vvar_n) :- c^j_i(Vvar_1,\ldots,Vvar_n).$<br>$lc^j_i(Vvar_1,\ldots,Vvar_n) :- lc^j_{i+1}(Vvar_1,\ldots,Vvar_n).$<br>$\mid 1 \leq i \leq n, Op_i = \mathbf{or}\}$<br>$\cup_{\{c^j_i \mid C^j \equiv c^j_1 Op_1 \ldots, Op_n c^j_n\}} \{\mathcal{C}^j_i\}$ | *(5)* $\{\$var_1,\ldots,\$var_n\} = DocVars(\Gamma).$ |
| $\mathcal{C}^j_i \equiv c^j_i(vvar_1(\overline{Tag^1}),\ldots,vvar_n(\overline{Tag^n})) :- Op(Tag^k_j, value).$<br>*whenever* $c^j_i \equiv \$var'/xpfree_j\ Op\ value$<br>*and* $Root(\$var') = \$var_k$<br>$\mathcal{C}^j_i \equiv c^j_i(vvar(\overline{Tag^1}),\ldots,vvar(\overline{Tag^n})) :- Op(Tag^k_j, Tag^m_r).$<br>*whenever* $c^j_i \equiv \$var'/xpfree_j\ Op\ \$var'/xpfree_r,$<br>$Root(\$var') = \$var_k$ *and* $Root(\$var') = \$var_m$ *(6)* | *(6)* $\{\$var_1,\ldots,\$var_n\} = DocVars(\Gamma).$ |
| $htag(\$var/xpfree_j) =_{def} join$<br>$tagpos(\$var/xpfree_j) =_{def} j$ *(7)* | *(7) for every* $\$var \in Vars(\Gamma)$, $xpfree_j \in Rootedby(\$var, \mathcal{X}) \cup Rootedby(\$var,\Gamma)$ |

Basically, the translation of *XQuery* expressions produces new rules (in the example *mybook*) having the form of "*views*" in which a *"join"* of documents is achieved (the *join* predicate makes the join). The join combines the values for *local variables whose value is the root of the input documents* (in the example \$*book* whose value is computed by *vbook*). The join also takes into account the constraints on these local variables (predicate *constraints*). Finally, for these local variables the set of required paths is computed. In the example, there is a local variable \$*book* whose value is the root of the document, and *title* and *year* are the required paths computed by *vbook*.

Now, we can build the goal for obtaining the answer for *xquery* as follows. Taking $htag(xquery) = mybook$ and $tagpos(xquery) = 1$ then the goal is $:- mybook(MyBook, Node, Type, Doc)$ and the answer is:

$MyBook = mybooktype(''XML\ in\ Scottland'', [''2002'']),\ Node = [[[1, 2], [1, 2]]$
$Type = [[[3, 3]]],\ Doc = [[''books.xml'']]$

This answer represents the XML document:

| **Answer as an XML document** |
| --- |
| *<mybook year="2002">* |
| *<title>XML in Scotland</title>* |
| *</mybook>* |

Let us remark that the output document is not numbered as the source documents. The join of several documents with different node and type numbering produces an unique output document. However, the output document is still indexed and typed by considering the list of node and type numbers of the input documents. In the example the first $[1, 2]$ represents the node number of the book titles, and the second $[1, 2]$ represents the node number of the book years. Analogously, the first "3" represents the type number of book titles and the second "3" the type number of book years. The numbering of output documents still allows the recovering of the hierarchical structure by considering the lexicographic order in lists. Due to the lack of space we omit here the details about the reconstruction of output documents.

## 5 Conclusions and Future Work

In this paper, we have studied how to translate *XQuery* expressions into logic programming. It allow us to evaluate *XQuery* expressions against XML documents using logic rules. As future work we would like to implement our technique. We have already implemented *XPath* in logic programming (see `http://indalog.ual.es/Xindalog`). Taking as basis this implementation we would like to extend it to *XQuery* expressions.

## References


[ABE06] J. M. Almendros-Jiménez, A. Becerra-Terón, and Francisco J. Enciso-Baños. Magic sets for the XPath language. *Journal of Universal Computer Science*, 12(11):1651–1678, 2006.

[ABE07] J. M. Almendros-Jiménez, A. Becerra-Terón, and Francisco J. Enciso-Baños. Querying XML documents in logic programming. *To appear in Theory and Practice of Logic Programming, available at `http://www.ual.es/~jalmen`*, 2007.

[BBFS05] James Bailey, Franois Bry, Tim Furche, and Sebastian Schaffert. Web and Semantic Web Query Languages: A Survey. In *Reasoning Web, First International Summer School*, volume 3564, pages 35–133. LNCS, 2005.

[BCF05] Veronique Benzaken, Giuseppe Castagna, and Alain Frish. CDuce: an XML-centric general-purpose language. In *Procs of the ACM ICFP*, pages 51–63. ACM Press, 2005.

[BGvK+05] Peter A. Boncz, Torsten Grust, Maurice van Keulen, Stefan Manegold, Jan Rittinger, and Jens Teubner. Pathfinder: XQuery - The Relational Way. In *Procs. of the VLDB*, pages 1322–1325. ACM Press, 2005.

[Bol00] H. Boley. Relationships between logic programming and XML. In *Proceedings of the Workshop on Logic Programming*, 2000.

[Bol01] H. Boley. The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations. In *Proc. of International Conference on Applications of Prolog, INAP*, pages 124–139. Prolog Association of Japan, 2001.

[CDF+04] D. Chamberlin, Denise Draper, Mary Fernández, Michael Kay, Jonathan Robie, Michael Rys, Jerome Simeon, Jim Tivy, and Philip Wadler. *XQuery from the Experts*. Addison Wesley, 2004.

[CF03] Jorge Coelho and Mario Florido. Type-based XML processing in logic programming. In *Proceedings of the PADL 2003*, pages 273–285. LNCS 2562, 2003.

[CH01] D. Cabeza and M. Hermenegildo. Distributed WWW Programming using (Ciao-)Prolog and the PiLLoW Library. *TPLP*, 1(3):251–282, 2001.

[Cha02] D. Chamberlin. XQuery: An XML Query Language. *IBM Systems Journal*, 41(4):597–615, 2002.

[HP03] H. Hosoya and B. C. Pierce. XDuce: A Statically Typed XML Processing Language. *ACM Transactions on Internet Technology, TOIT*, 3(2):117–148, 2003.

[May04] W. May. XPath-Logic and XPathLog: A Logic-Programming Style XML Data Manipulation Language. *Theory and Practice of Logic Programming, TPLP*, 4(3):239–287, 2004.

[MS03] A. Marian and J. Simeon. Projecting XML Documents. In *Procs. of VLDB*, pages 213–224. Morgan Kaufmann, 2003.

[OOP+04] Patrick O'Neil, Elizabeth O'Neil, Shankar Pal, Istvan Cseri, Gideon Schaller, and Nigel Westbury. ORDPATHs: insert-friendly XML node labels. In *Procs. of the ACM SIGMOD*, pages 903 – 908. ACM Press, 2004.

[SB02] S. Schaffert and F. Bry. A Gentle Introduction to Xcerpt, a Rule-based Query and Transformation Language for XML. In *Proc. of RuleML*, 2002.

[Sei02] D. Seipel. Processing XML-Documents in Prolog. In *Procs. of the Workshop on Logic Programming 2002*, 2002.

[Thi02] Peter Thiemann. A typed representation for HTML and XML documents in Haskell. *Journal of Functional Programming*, 12(4&5):435–468, 2002.

[TVB+02] Igor Tatarinov, Stratis D. Viglas, Kevin Beyer, Jayavel Shanmugasundaram, Eugene Shekita, and Chun Zhang. Storing and querying ordered xml using a relational database system. In *Procs of the ACM SIGMOD*, pages 204–215. ACM Press, 2002.

[W3C07a] W3C. XML Path language (XPath) 2.0. Technical report, www.w3.org, 2007.

[W3C07b] W3C. XQuery 1.0: An XML Query Language. Technical report, www.w3.org, 2007.

[Wad02] P. Wadler. XQuery: A Typed Functional Language for Querying XML. In *Advanced Functional Programming, 4th International School, AFP*, LNCS 2638, pages 188–212. Springer, 2002.

[Wie05] J. Wielemaker. SWI-Prolog SGML/XML Parser, Version 2.0.5. Technical report, Human Computer-Studies (HCS), University of Amsterdam, March 2005.

[WR99] Malcolm Wallace and Colin Runciman. Haskell and XML: Generic combinators or type-based translation? In *Proceedings of the International Conference on Functional Programming*, pages 148–159. ACM Press, 1999.