JKAU: Eng. Sci., **Vol. 21** No.2, pp: 35-57 (1431A.H./2010 A.D.)
DOI: 10.4197 / Eng. 21-2.3

A Survey of a Family of Recursive Query Languages for XML Data

Mourad Ykhlef
College of Computer and Information Science, King Saud University, Riyadh, Saudi Arabia
ykhlef@ksu.edu.sa

Abstract. XML is fast emerging as the dominant standard for information exchange on the World Wide Web. The ability to intelligently query XML data becomes increasingly important. Several query languages are proposed in the literature, they are mainly different in the underlying model and in the power of expression. This article present and compare a family of recursive query languages for querying and restructuring XML data.

Keywords: XML, Tree, Embedding, GROUP BY, Fixpoint, (Structural) recursion.

1. Introduction

XML, the extensible Markup Language, has rapidly grown as a standard for information exchange on the World Wide Web. XML data are data whose structure is not necessary constrained by a schema[1]. The need of XML query languages is justified by the increasing data published in XML data format and the necessity of exploiting these huge amounts of data. Designing query languages for XML data has received a lot of attention during the last few years. Unlike relational query language SQL, the design of XML query languages is more complicated. In SQL the result is not reconstructed in the syntax, the result is always a table. However the XML query languages have generally two parts: querying part and result reconstructing part. The XML query languages have to intermix both parts in a nested way and thus make the querying more

difficult. XML query languages have also to integrate recursion for augmenting their power of expression. Without recursion mechanism one cannot express the query finding a path in an XML network data.

In this article we present a family of XML recursive query language: XRQL[2], XML-RL[3,4], XQuery[1,5] and UnCAL[6,7,8] for querying and restructuring XML data. The common characteristic of these languages is the presence of recursion. XRQL is SQL like query language which integrates recursion and grouping. XML-RL is a rule based query language integrating recursion and grouping. XQuery is a language normally designed for querying but it is rather a programming language than querying language because of the intensive use of programming constructs like FOR and recursive functions. UnCAL is calculus query language integrating structural recursion[9,10] in its formalism.

The remainder of this article is organized as follows: Section 2 presents XML data models. Section 3 is devoted to the presentation of XRQL. However in Sections 4, 5 and 6 we present XML-RL, XQuery and UnCAL respectively. A comparison of these query languages is given in section 7.

2. XML Data Modeling

2.1 *G-XML Model*

In G-XM model[2], an XML document is modeled by a labeled rooted graph called G-XML where:

- each edge is labeled by an XML element or XML attribute.
- each leaf node is labeled with a value.
- G-XML has a distinguished node called the root.

For example, the XML document of Fig. 1, giving information on bibliographic data (bib.xml), is represented by the G-XML graph of Fig. 2. Remark that the edge labeled by `title` represents an element tag but the edge labeled by `year` represents an attribute tag. Attributes are alternative ways to represent data. It is important to note that object identifiers (oids) and references in XML document are just syntax. The G-XML graph representing XML data, in presence of oids and references, can be done straightforwardly. G-XML graphs are not only

derived from XML document, they can be also generated by XRQL nested queries[2].

```
<bib>
<book year="1988">
  <title>Principles of Database and Knowledge Base
  Systems</title>
  <author><name>Jeff Ullman</name></author>
  <publisher>W.H. Freeman Company</publisher>
  <isbn>0716781581</isbn>
</book>
<book year="1995">
  <title>Foundations of Databases</title>
  <author>
    <name>Serger Abiteboul</name>
  </author>
  <author><name>Rick Hull</name></author>
  <author><name>Victor Vianu</name></author>
  <publisher>Addison-Wesley</publisher>
  <isbn>0201537710</isbn>
</book>
<article>
  <title>Querying Semi-structured Data</title>
  <author>
    <name>Serge Abiteboul</name>
  </author>
  <proceedings>ICDT</proceedings>
</article>
</bib>
```

Fig. 1. XML bibliographic data.

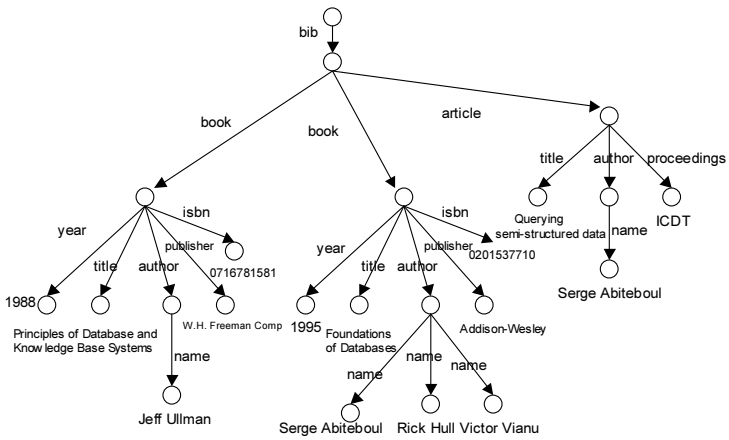

Fig. 2. A G-XML graph for bibliographic data.

2.2 *XML Data Versus Complex Object*

In Complex Object Model[3], the XML data are viewed as a complex object, with such view; query representation becomes also higher level as well. The XML data of Fig. 1 are represented in this model as it is depicted in Fig. 3.

```
bib=>
[book=>[@year=>1988,
        title=>Principles of Database and Knowledge Base
        Systems
        author=>[name=>Jeff Ullman]
        publisher=>W.H. Freeman Company
        isbn=>071678158]
book=>[@year=>1995,
      title=>Foundations of Databases
      author=>[name>=>Serge Abiteboul]
      author=>[name>=>Rick Hull]
      author=>[name>=>Victor Vianu]
      publisher=>Addison-Wesley
      isbn=>0201537710]
article=>[title=>Querying Semistructured Data
         author=>[name>=>Serge Abiteboul]
         proceedings=>ICDT]]
```

Fig. 3. A regular representation of bibliographic data.

2.3 *Query Data Model*

The Query Data Model[1] is a form of tree representation. The Query Data Model is based on the notion of a sequence. A sequence is an ordered collection of zero or more items. An item may be a node or an atomic value. An atomic value is an instance of one of the built-in data types defined by XML schema, such as strings, integers, decimals and dates. A node conforms to one of seven node kinds, which include element, attribute, text, document, comment, processing instruction, and namespace nodes. A node may have other nodes as children, thus forming one or more node hierarchies. Some kinds of nodes, such as element and attribute nodes, have names or typed values, or both. A typed value is a sequence of zero or more atomic values. Nodes have identity (that is, two nodes may be distinguishable even though their names and values are the same), but atomic values do not have identity. Among all the nodes in a hierarchy there is a total ordering called document order, in which each node appears before its children. The XML data of Fig. 1 are partially represented in query data model as depicted in Fig. 4.

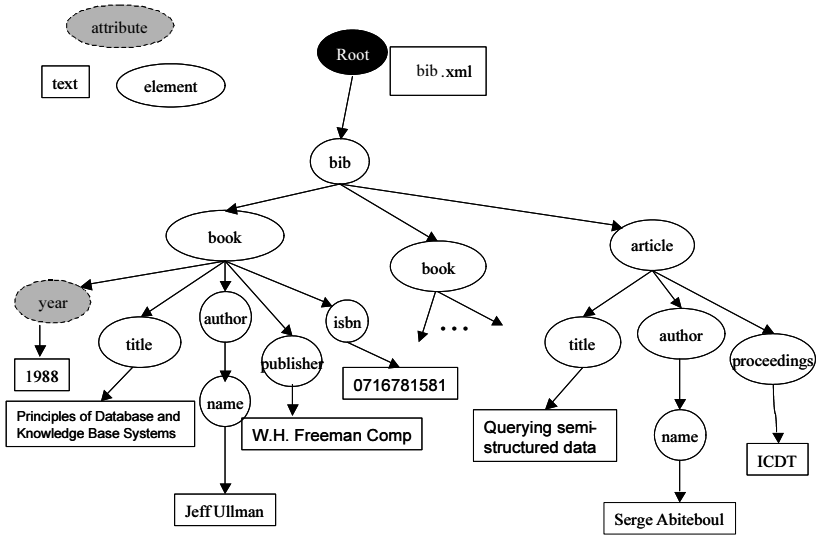

Fig. 4. A Query data model for bibliographic data.

2.4 *XML Data Versus Tree*

XML data can be represented, in tree model[6,7,8], by a tree (or a graph); labels are always on edges. A tree is defined recursively as follows:

```
t ::= a|{l:t,…,l:t}
```

where `a` ranges over the atomic values and `l` ranges over labels. The tree representation of XML data of Fig. 1 is given in Fig. 5.

```
{bib:{book:{year:1988,
            title:Principles of Database and Knowledge Base Systems,
            author:{name:Jeff Ullman},
            publisher:W.H. Freeman Company,
            isbn:071678158},
    book:{year:1995,
          title:Foundations of Databases,
          author:{name:Serger Abiteboul}
          author:{name:Rick Hull}
          author:{name:Victor Vianu}
          publisher:Addison-Wesley,
          isbn:0201537710},
    article:{title:Querying Semi-structured Data,
           author:{name:Serger Abiteboul},
           proceedings:ICDT}
}
```

Fig. 5. A Tree representation of bibliographic data.

3. Querying and Restructuring XML Data by XRQL

In this section we present XRQL[2], a recursive query language for querying and restructuring XML data represented by G-XML model (see Section 2.1). The main characteristics of XRQL are:

- the absence of programming constructs such as “for" used in XQuery language[1,5].
- XRQL has a Fixpoint operator allowing the expression of recursive queries in the same spirit of Graph-Fixpoint[11,12] and XML-RL[3,4]. The absence of the multiple occurrence symbol “*” from XRQL path expressions does not matter. The symbol “*” can be simulated by Fixpoint operator.
- XRQL allows the definition of intensional predicates over XML data in an elegant manner, as it will be illustrated in example 8 and example 9.
- XRQL allows XML data restructuring.
- XRQL provides grouping operator and aggregate functions.
- XRQL queries can be nested.

The language XRQL is based on path expressions, XRQL is very similar to a multi-sorted first order language. Because in G-XML data model, data are associated to leaves, XRQL path expressions may contain data variables as abstractions of the content of leaves. They may also contain label (resp. path) variables as abstraction of edges (resp. paths). The graph variables abstract sub-graphs in a G-XML data model.

3.1 *Path Expression*

In the following, it is assumed that four sorts of variable sets are given: a set of graph (resp. path, label, and data) variables. Graph and path variables are denoted X, Y, Z, ... Label (resp. data) variables are denoted x, y, z, …

In order to define path expression we need to define label term and data term. A label term is either an edge label or a label variable. A data term is either leaf value or data variable. A path is simple w.r.t edges if all its edges are distinct. In the rest of presentation a path refers always to a simple path. An empty path ε is a path with no edges.

Definition 1: [Path Expression] A path expression is a sequence of label terms and/or path variables.

If s is a path expression and t is a data term then s@t is a path expression. The path expression s@t means that any path captured by s must have a destination (end) equal to the value associated with the data term t.

Example 1: For example the path expression `book.author` is intended to represent all paths of length 2 having as edge labels the label `book` and then the label `author`. The path expression `book.x` is intended to represent all paths of length 2 where the first edge label is `book`; the label variable `x` is valuated to one label in the set of labels `{year, title, author, publisher, isbn}`. The path expression `X.author` represents each path having at least one edge where the last edge is labeled by `author`. The path variable `X` is valuated to the empty path ε or to a path of length 1 having `book` as edge label. The path variable `X` can be also valuated to the path of length 2 captured by the path expression `bib.book`. The data variable `x` in the path expression `book.title@x` captures all leaves values at the destination of paths abstracted by the path expression `book.title`.

Now some terms are defined. A path (resp. graph) term is either a (simple) path (resp. graph) or path (resp. graph) variable. Remember that a label term is either an edge label or a label variable. A data term is either leaf value or data variable. The origin (resp. destination) of a path is the first (resp. last) node. A node `r` of a graph `G` is a source for `G` if there exists a path from `r` to any other nodes of `G`.

Definition 2: [Atom] An atom is an expression having one of the following forms:

1. *a path expression,*
2. t1=t2 *where* t1 *and* t2 *are terms of the same sort among graph, path, label, data; = is the (polymorphic) equality predicate symbol,*
3. t=s *where t is a path term, s is a path expression and = is an equality predicate.*
4. t:s *where t is a graph term and s is a path expression.*
5. s[t] *where t is a graph term and s is a path expression.*

Intuitively, the atom `t=s` intends to check whether `t` is one of the paths captured by the path expression `s`. The atom `t:s` tells that `s` captures at least one path `p` of the graph `t` and that the origin of `p` is the

source of t; the intention of the atom s[t] is to check that the graph t has a source which is the destination of a path captured by the path expression s. For example, the atom X:book.title intends to capture all paths p having successively book and title as edge labels in the G-XML sub-graph representing the X valuation such that the origin of p is the source of X. However, the atom book.title[X] captures all sub-graphs X at the destination of paths captured by the path expression book.title.

3.2 *XRQL Query*

The syntax of XRQL query is SQL-like, it has the form SELECT FROM WHERE; where the clause SELECT contains XML document fragments with variables, the valuation of SELECT yields an XML document. The clause FROM specifies a series of XML documents to be queried; it has the following form:

FROM f_1.xml[AS D_1],...,f_n.xml[AS D_n]

where [AS D_i] is a renaming option, the graph variable Di is valuated to the G-XML graph associated to a file f_i.xml. When there is only one document in a query; one can drop the option [AS D_i].

Example 2: To return the title of all books, we write the following query:

```
SELECT <title> x </title> AS <books>
FROM bib.xml
WHERE book.title@x
```

Intuitively this query selects all book titles <title>x</title> and environs them by the element <books>. This query posing on the G-XML graph of Figure 2 returns the following XML data:

```
<books>
   <title>Principles of Database and Knowledge Base Systems</title>
   <title>Foundations of Databases</title>
</books>
```

The same query can be expressed as below:

```
SELECT P AS <books>
FROM bib.xml
```

```
WHERE book.P and P=title
```

Note that `P` is a path variable and the atom `P=title` intends to check whether `P` is one of the paths captured by the path expression title.

As we have previously said the clause `SELECT` can contain the expression `AS <elem>`. The element `elem` environs the last collection of elements returning before `AS`. Note that in the expression

```
P Q AS <elem>
```

where `P` and `Q` are path and/or graph variables, `<elem>` environs only the collection of elements captured by `Q`.

Example 3: The below query returns all articles written by `Serge Abiteboul` and appeared in *`ICDT`* proceedings:

```
SELECT <article>X</article> AS <articles>
FROM bib.xml
WHERE bib.article[X] and
      X:author.name@"Serge Abiteboul" and
      X:proceedings@"ICDT"
```

The atom `bib.article[X]` intends to check whether the graph variable `X` has a source which is the destination of a path captured by the path expression `bib.article`. The atom

```
X:author.name@"Serge Abiteboul"
```

tells that the path expression `author.name@"Serge Abiteboul"` captures at least one path `p` of the graph `X` and that the origin of `p` is the source of `X`. The result of posing the above query on the G-XML graph of Fig. 2 is:

```
<articles>
  <article>
    <title>Querying Semi-structured Data</title>
    <author><name>Serge Abiteboul</name></author>
    <proceedings>ICDT</proceedings>
  </article>
</articles>
```

Example 4: The following query returns authors who have written books and articles:

```
SELECT <name>x</name> AS<authors>
FROM bib.xml
WHERE  article.author.name@x and
       book.author.name@x
```

Example 5: [Data integration] Let us consider the G-XML graph of Fig. 2 and let us consider the following XML data file (`yellow.xml`) giving addresses and phone numbers of people:

```
<persons>
  <person>
    <name>Serge Abiteboul</name>
    <address>INRIA-France</address>
    <phone>+33 1 72 92 59 18</phone>
  </person>
</persons>
```

The following query returns the address and the phone number of each article author:

```
SELECT <author>
         <name>x</name>
         <address>y</address>
         <phone>z</phone>
       </author> AS <result>
FROM bib.xml AS D1, yellow.xml AS D2
WHERE D1:bib.article.author.name@x
      and D2:persons.person[Y]
      and Y:name@x and Y:address@y
      and Y:phone@z
```

The execution result is:

```
<result>
  <author>
    <name>Serge Abiteboul</name>
    <address>INRIA-France</address>
    <phone>+33 1 72 92 59 18</phone>
  </author>
</result>
```

3.3 *Grouping*

In this section we present two kinds of grouping: grouping by embedding and grouping by using `GROUP BY` operator.

3.3.1 *Grouping by Embedding*

One way of doing grouping is to embed XRQL queries into XML data as it is illustrated in the following example.

Example 6: For each author, return the list of his/her publications:

```
SELECT <authortitles>
          P1
          SELECT Q AS <titles>
          FROM bib.xml
          WHERE (book[X] or article[X])
            and X:Q and Q=title
            and X:P2 and P2=author.name
            and P1=P2
       </authortitles> AS <result>
FROM bib.xml
WHERE P1=author.name
```

Note that XML data returning by the first `SELECT` contains a nested query. Posing this query on the G-XML graph of Fig. 2 will return, among others, the following XML data:

```
<result>
  ...
  <authortitles>
    <author><name>Serge Abiteboul</name></author>
    <titles>
      <title>Foundations of Database</title>
      <title>Querying Semi-structured Data</title>
    </titles>
  </authortitles>
  ...
</result>
```

3.3.2 *Grouping by GROUP BY Operator*

The `GROUP BY` statement has the following form:

```
SELECT ... var1 [AS <element1>]
```

```
       ... var_n [AS <element_n>] ...
FROM ...
WHERE ...
GROUP BY var_1,...,var_n
```

where variables $var_1,\ldots,var_n$ are either path or graph variables and the symbol `[]` indicates that element is optional. In the presence of aggregate functions like `COUNT, MAX, MIN, AVG, SUM`, references in the `SELECT` clause can only be made to aggregate functions and to the variables in the `GROUP BY` clause. However, in the absence of aggregate functions, all variables in `SELECT` clause must appear in `GROUP BY`.

Example 6: **(Continued)** The same query can be reformulated by using `GROUP BY` as follows:

```
SELECT <authortitles>
        P Q AS <titles>
       </authortitles> AS <result>
FROM bib.xml
WHERE (book[X] or article[X])
       and X:P and P=author.name
       and X:Q and Q=title
GROUP BY P,Q
```

XRQL can also use `HAVING` with `GROUP BY` operator to return, for example, authors with publication number above a certain number as it is illustrated in the following example.

Example 7: The following query returns authors with publication number above one:

```
SELECT <authortitles>
         P <numpub>COUNT(Q)</numpub>
        </authortitles> AS <result>
FROM biblio.xml
WHERE (book[X] or article[X])
       and X:P and P=author.name
       and X:Q and Q=title
GROUP BY P
HAVING COUNT(Q)>1
```

3.4 *Recursion*

We first begin by presenting transitive closure, which is a simple case of recursion, and then a more complex example of recursion is given.

Example 8: **(Transitive closure)** Let us consider the XML data `edge.xml` representing the relation `edge(departure,arrival)`:

```
<db>
  <edge>
    <departure>a</departure>
    <arrival>b</arrival>
  </edge>
  <edge>
    <departure>b</departure>
    <arrival>c</arrival>
  </edge>
  <edge>
    <departure>c</departure>
    <arrival>d</arrival>
  </edge>
</db>
```

The following query returns the paths existing in the above XML data.

```
1. WITH TC(P,Q)
2.    SELECT P,Q
3.    FROM edge.xml
4.    WHERE edge[X] and X:P and X:Q
            and P=departure
            and Q=arrival
5.  union
6.    SELECT P,Q
7.    FROM edge.xml
8.    WHERE edge[X] and X:P and X:R
            and P=departure
            and R=arrival
            and TC(S,Q)
            and R@x and S@x
```

Note that XRQL queries are restricted here to return tuples of paths. The above query shows how to compute the predicate **TC**. Line (1) introduces the definition of **TC**, while the real definition of this predicate is in line (2) through (8). That definition is a union of two queries. The first query (line (2) through (4)) is a basis case of the recursion, however the second query (line (6) through (8)) represents the recursion. Note that the expression **TC**(S,Q) means that the couple of path variables S and Q is one tuple among the set of TC tuples. The formula R@x and S@x checks whether the paths captured by the path expressions R and S have the same destination.

Example 9: **[Equal length path]** The following query returns the paths of the same length.

```
WITH SL(P,Q)
   SELECT P,Q
   FROM anyfile.xml
   WHERE P=ε and Q=ε
  union
   SELECT P,Q
   FROM anyfile.xml
   WHERE SL(X,Y) and P=X.x and Q=Y.y
```

In the basic case of the recursion we set (P,Q) equal to the tuple of empty paths (ε,ε). Note that this query does not make use of any predefined counting predicate, this query is rather important for sequence data. In the same spirit of this query, one can express a recursive query that extracts all paths generated for example by the regular language $\{a^n b^n \mid n \in \mathrm{N}\}$.

4. XML-RL Language

The language XML-RL was proposed by Liu, M.[3,4], the underlying model is based on complex object presented in Section 2.2.

4.1 *Basic Queries*

The query of example 2 returning the title of all books, is written in XML-RL as follows:

```
 /books⇒[title⇒$t]
⇐
```

```
/bib/book⇒[title⇒$t]
```

Note that the head of the rule (*i.e.*, expression before ⇐) is a result constructor.

The query of example 3, which provides all `Serge Abiteboul` articles appeared in `ICDT` proceedings, is given by:

```
/article⇒[title⇒$t]
⇐
/bib/article⇒[title⇒$t,
              /author⇒[name⇒"Serge Abiteboul"],
              proceedings⇒"ICDT"]
```

However the query of example 4, returning authors who have written books and articles, is expressed as follows:

```
/authors⇒[name⇒$a]
⇐
/bib/book/author⇒[name⇒$a],
/bib/article/author⇒[name⇒$a]
```

4.2 *Grouping*

The query of example 6 returning the list of publications of each author, can be expressed in XML-RL as follows:

```
(file://temp.xml)/titles/authortitles⇒
                  [author⇒$a, title⇒{$t}]
⇐
/bib/book⇒[title⇒$t, /author/name⇒$a]

(file://temp.xml)/titles/authortitles⇒
                  [author⇒$a, title⇒{$t}]
⇐
/bib/article⇒[title⇒$t, /author/name⇒$a]
```

The grouping variable `{$t}` in the rule head (construction part) is used to group the titles of all books or articles by the author `$a`. The query result is stored in `temp.xml` file.

The query of example 7, returning authors with publication number above 1, cannot be expressed directly. One should first write the previous

query to produce the file `temp.xml` which can be queried by the following query:

```
 (file://temp.xml)/titles/authortitles⇒
                    [author⇒$a, title⇒{$t}]
⇐
 (file://temp.xml)/titles/authortitles⇒
                    [author⇒$a, title#t],
                    count(#t)>1
```

Note here that the variable `#t` is a list-valued variable that match a list of publications (books or articles).

4.3 *Recursion*

The transitive closure of the example 8 can be formulated in XML-RL as follows:

```
 (file://tc.xml)/results/path⇒
                [departure⇒$d, arrival⇒$a]
⇐
 /db/edge⇒[departure⇒$d, arrival⇒$a]

 (file://tc.xml)/results/path⇒
               [departure⇒$d, arrival⇒$a]
⇐
 /db/edge⇒[departure⇒$d, arrival⇒$z],
 (file://tc.xml)/results/path⇒
               [departure⇒$z, arrival⇒$a]
```

However the query of example 9 cannot be expressed because XML-RL has no Fixpoint operator on paths.

5. XQuery Language

The query language XQuery[1,5] is an XML query language developed by the XML Query Working Group at the World Wide Web Consortium. XQuery is derived from a query language called Quilt[13], the underlying model called Query data model is presented in section 2.3.

5.1 *Basic Queries*

The query of example 2, returning the title of all books, is written in XQuery as follows:

```
<books>{
  for $b in doc(bib.xml)/bib/book
  return
    {$b/title}
}</books>
```

where `for` is a loop symbol and the variable `$b` ranges over the result sequence, getting one item at a time.

The query of example 3, providing all `Serge Abiteboul` articles appeared in `ICDT` proceedings, is expressed as follows:

```
<articles>{
  for $b in doc(bib.xml)/bib
  let $p := $b/article
  where
    $p/author/name="Serge Abiteboul" and
    $p/proceedings="ICDT"
    return
      {$b/article}
}</articles>
```

Note that `let` is an assignment symbol, `$p` gets the entire result of `$b=article` (possibly many nodes). The symbol `where` is a filter condition.

The query of example 4, returning authors who have written books and articles, is expressed as follows:

```
<authors>{
  for $b1 in doc(bib.xml)/bib/book
  for $b2 in doc(bib.xml)/bib/article
  where $b1/author = $b2/author
  return
    {$b1/author}
}</authors>
```

5.2 *Grouping and Recursion*

The query of example 6, returning for each author the list of his/her publications, can be expressed in XQuery by embedding `for` into another `for`. Beyer, K., *et al*.[14] have added an explicit `GROUP BY` operator to XQuery to express grouping without embedding. Concerning

recursive queries; the transitive closure of the example 8 cannot be expressed in standard XQuery. However XQuery augmented with recursive functions allows one to express recursion. XQuery, augmented with recursive functions and other programming constructs such as `FOR`, tends to be a programming language rather than querying language. The language XQueryP[15] is a programming version of XQuery.

6. UnCAL Query Language

The UnCAL (Unstructured CALculus)[6,7,8] is a lambda calculus based query language. The underlying model is defined in section 2.4. UnCAL integrates particularly the structural recursion[9,10], the tree traversal is guided by the structure. The language UnCAL uses three forms of recursive functions to traverse a tree: `ext`, `text` and `text'`.

The first function takes consideration only of the first level of a tree. The second one traverses the whole tree without any stops condition however the last one makes a conditional traversal (*i.e.*, stops in the recursion are possible).

1. Given a function `f:label x tree → tree`, `ext` applies `f` in the following way:

```
ext({}) = {}
ext({l:t}) = f(l,t)
ext(t1∪t2) = ext(t1)∪ext(t2)
```

For example, if `f(l,t)` is of the form

```
if l = bib then t else {}
```

the function `ext`, applied on the tree of Fig. 5, returns three sub-trees rooted at the edge labeled by `bib`.

2. Given a function `f:label x tree → tree`, `text` applies `f` in the following way:

```
text({}) = {}
text({l:t}) = f(l,text(t))
text(t1∪t2) = text(t1)∪text(t2)
```

For example, if `f(l,text(t))` is of the form `l∪text(t)`, then the function `text`, applied on the tree of Fig. 5, returns the flat tree:

`{bib,book,article,year,title,author, name,…}`

The composition of `ext` and `text` given by:

```
                text({}) = {}
          text({l:t}) = {l}∪ext(text(t))
       text(t1∪t2 ) = text(t1 )∪text(t2)

                ext({}) = {}
           ext({l1:t}) = {l:{l1 :t}}
        ext(t1∪t2) = ext(t1)∪ext(t2)
```

when applied on the tree of Fig. 5 returns the set of all (simple) paths starting from the root.

```
{bib,
bib : book,
bib : {book : year},
bib : {book : {year : 1988}},
…
bib : {article : {proceedings : ICDT}}}
```

This example is very interesting because it illustrates how some queries needing path variable, may be expressed in UnCAL.

3. Given a function `f:label x tree x tree → tree`, `text'` applies `f` in the following way:

```
               text'({}) = {}
         text'({l:t}) = f(l,t,text'(t))
      text'(t1∪t2) = text'(t1)∪text'(t2)
```

If `f(l,t,text'(t))` is of the form `{l:t}∪text'(t)` then the function `text'`, applied on the tree of Fig. 5, returns all sub-trees.

The structural recursion has two major advantages: (i) its semantics is clear on trees with or without cycles (ii) it is executed in polynomial time. In order to handle cycles, the operational semantics of UnCAL consists to memorize the recursive calls to avoid infinite loops. The structural recursion offers the possibilities to develop more sophisticated optimization techniques[7]. Obviously, UnCAL is not user-friend

language. It was designed to support optimization of the tree traversals. Buneman, P., *et al*.[6] have proposed a new SQL-like query language. Its semantics is based on UnCAL. UnCAL can itself be captured using structural recursion.

6.1 *Grouping and Recursion*

The grouping is expressed in UnCAL via query embedding as XRQL did in Section 3.3.1. Concerning recursive queries, the transitive closure of the example 8 cannot be expressed in UnCAL. The power expression of UnCAL on relational data represented by a tree is that of first order logic.

However, UnCAL can express some forms of transitive closure, namely by navigating the graph, *e.g*. through a regular expression[1].

7. Conclusion

In this article, we have presented a family of recursive query languages for querying and restructuring XML Data. In Table 1, we give a general comparison of these languages in terms of data model used, style, grouping and expressive power. The power of expression is given in terms of classes of relational query posed on relational data modeled by a tree. FO stands for First Order queries and TC stands for Transitive Closure. The Fixpoint queries express more general form of recursion than TC queries[16]. It is important to note that although XQuery is in FO+Fixpoint but it tends to be a programming language rather than a querying language. XQuery uses the programming constructs like `FOR` and recursive functions. The language XRQL is the most interesting query language because of the natural use of recursion and grouping. XRQL did not use any programming constructs used by XQuery. The query language XML-RL is also an important language but it lacks a natural way for data grouping.

(1) See the even/odd example on page 94 of the paper [6]

Table 1. Comparison of XML query languages.

Language	Data model	Style	Grouping	Recursion	Expressive power
XRQL	Graph	SQL	Embedding or Explicit use of GROUP BY	Yes	FO+ Fixpoint
XML-RL	complex object	Rule	Not Explicit	Yes	FO+ Fixpoint
XQuery	Graph	SQL	Embedding of FOR	NO	FO
XQuery+ recursive fun	Graph	SQL+ programming constructs	Embedding of FOR	Yes	FO+ Fixpoint
UnCAL	Tree	lambda calculus	Embedding	NO	FO

References

[1] **Chamberlin, D.,** "XQuery: An XML query language," *IBM Systems Journal*, **41**(4):597-615, (2002).

[2] **Ykhlef, M.,** "Recursive SQL-like Query Language for XML," *The 9th Int. Conf on Information Integration and Web-based Applications and Services (IIWAS 2007), Jakarta, Indonesia, 3-5 Dec* (2007).

[3] **Liu, M.,** "*A Logical Foundation for XML".* CAISE 2002, volume 2348 of Lecture Notes in Computer Science, pages 568-583, Springer-Verlag, (2002).

[4] **Liu, M.** and **Wang Ling, T.,** "*Towards Declarative XML Querying*," WISE, pp:127-138. IEEE Computer Society (2002).

[5] **Chamberlin, D.** and **others.,** "*XQuery: A query language for XML*," http://www.w3.org/TR/2001/WD-xquery-20010215, Februry (2001).

[6] **Buneman, P., Fernandez, M.F.** and **Suciu, D.,** "UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion**"**, *VLDB J.,* **9**(1):76-110 (2000).

[7] **Buneman, P., Davidson, S.B., Hillebrand, G.** and **Suciu, D.,** "A Query Language and Optimization Techniques for Unstructured Data", *SIGMOD Conference*, pp: 505-516. ACM Press (1996).

[8] **Buneman, P., Davidson, S.B.** and **Suciu, D.,** "Programming Constructs for Unstructured Data". Paolo Atzeni and Val Tannen, ed., *DBPL, Electronic Workshops in Computing, page 12. Springe* (1995).

[9] **Breazu-Tannen, V., Buneman, P.** and **Naqvi, S.,** "Structural Recursion as a Query Language*". Proceedings of International Workshop on Database Programming Languages, pages 9-19, Nafplion, Greece* (1991).

[10] **Breazu-Tannen, V., Buneman, P.** and **Wong, L.,** "Naturally Embedded Query Languages", *Proceedings of International Conference on Database Theory (ICDT),* pp: 140-154, Berlin, Germany, October (1992).

[11] **Bidoit, N.** and **Ykhlef, M.,** "Fixpoint Path Queries", International Workshop on the Web and Databases WebDB'98, *Conjunction with EDBT'98*, pp: 56-62, Valencia, Spain, 27-28 March (1998).

[12] **Bidoit, N.** and **Ykhlef, M.,** "Fixpoint Calculus for Querying Semistructured Data". In The World Wide Web and Databases, *Int. Workshop WebDB'98*, *Spain,* Selected Papers, volume **1590** of Lecture Notes in Computer Science, pp: 78-97, Springer-Verlag (1999).

[13] **Chamberlin, D., Robie, J.** and **Florescu, F.,** "Quilt: An XML Query Language for Heterogeneous Data Sources", In Dan Suciu and Gottfried Vossen, ed., WebDB (Selected Papers), volume **1997** of *Lecture Notes in Computer Science,* pp: 1-25. Springer (2000).

[14] **Beyer, K., Chamberlin, D., Colby, L., Ä Ozcan, F., Pirahesh, H.** and **Xu, Y.,** "Extending XQuery for Analytics". In Fatma Ä Ozcan, editor, *SIGMOD Conference*, pp:503-514. ACM, (2005).

[15] **Chamberlin, D., Carey, M. , Florescu, D., Kossmann, D.** and **Robie, J.,** "XQueryP: Programming With XQuery". *Third International Workshop on XQuery Implementation, Experience, and Perspectives, Chicago* (2006).

[16] **Abiteboul, S.** and **Vianu, V.,** "Computing with first-order logic", *J. Computer System Science.,* **50**(2):309-335 (1995).

دراسة استقصائية لأسرة من لغات الاستعلام التكرارية لبيانات اكسمل

مراد يخلف

كلية علوم الحاسب والمعلومات– جامعة الملك سعود – الرياض، المملكة العربية السعودية

المستخلص. لقد بزغ اكسمل بسرعة كمعيار مهيمن في مجال تبادل المعلومات على الشبكة العنكبوتية العالمية. إن القدرة على الاستعلام الذكي لبيانات اكسمل أصبحت ذات أهمية متزايدة. هناك عدة لغات استعلام مقترحة في أدبيات الموضوع، تختلف هاته اللغات فيم بينها في النموذج الكامن وراءها وفي قوة التعبير. سنقوم، في هذا البحث، بدراسة استقصائية مقارنة لأسرة من لغات الاستعلام التكرارية الموجهة لاستعلام وهيكلة بيانات اكسمل.

الكلمات الدالة: اكسمل، شجرة، التضمين، قروب – بي، النقطة الثابتة، التكرارية البنيوية.