An expressive and efficient language for XML information retrieval

Taurai Chinenyanga   Nicholas Kushmerick
Smart Media Institute, Department of Computer Science, University College Dublin
{taurai.chinenyanga, nick}@ucd.ie
www.smi.ucd.ie/elixir

Existing XML query languages do not support ranked query results based on textual similarity. For example, Fig. 1 shows an XML database containing books and CDs. We are interested in information-retrieval-style queries such as "order the items by similarity to the phrase 'traditional Ukrainian"' or "find books and CDs with similar titles.

Unlike similar efforts, our **e**xpressive and **e**fficient **l**anguage for **X**ML **i**nformation **r**etrieval (ELIXIR) allows such queries. ELIXIR extends XML-QL [3] with a textual similarity operator. Fig. 1 shows an ELIXIR query Q_1 that finds books and CDs with similar titles. Some related query languages (*e.g.* [5, 7]) provide only boolean keyword filtering, not ranked retrieval based on textual similarity. Other languages (*e.g.* [9, 6]) permit similarity comparisons only between a data value and a constant, but not similarity joins across two data values, and thus cannot express Q_1.

A more complicated ELIXIR query is shown in Fig. 2. As can be seen from the output, this query finds recent SIGMOD Record publications, New Testament verses, and lines from Macbeth that are similar.

Similarity joins are not merely of theoretical interest. For example, in data integration applications involving reconciling heterogeneous textual identifiers, similarity joins can eliminate the need for either common domains or hand-crafted normalization routines.

A naive implementation of a similarity join between two variables would generate the full cross product of the variable bindings, and then compute the similarity of every pair. Our ELIXIR query processing algorithm avoids this pitfall by:

1. rewriting an ELIXIR query Q_1 into a series of XML-QL Q_2^i queries that generate intermediate relational data;
2. invoking WHIRL [2] to efficiently evaluate Q_1's similarity predicates on this intermediate data; and
3. translating WHIRL's output into the XML structure specified by Q_1 with a final XML-QL query Q_4.

Note that ELIXIR queries XML data in its native format, without the potentially expensive operation of flattening it into relations [4, 8].

Fig. 1 shows how the ELIXIR query processing would rewrite Q_1 into a series of XML-QL queries Q_2^i, a WHIRL query Q_3, and a final XML-QL query Q_4. Also shown are the intermediate data that would be passed between the queries, and the final XML output. The algorithm first generates two XML-QL Q_2^i queries. Q_2^1 retrieves books from the XML database, and Q_2^2 retrieves CDs. Note that ELIXIR retrieves these items separately to avoid computing the cross product of book/CD pairs. The algorithm then generates the WHIRL query Q_3, which applies Q_1's similarity predicate. Q_3's output is the desired result, but it is transformed into XML format by executing Q_4.

ELIXIR's query rewriting algorithm runs in time polynomial in the size of the query, and thus ELIXIR's additional functionality adds negligible overhead to the cost of the underlying XML-QL and WHIRL implementations. Experiments with our ELIXIR prototype demonstrate that our query processing algorithm scales reasonably well with respect to the size and complexity of an ELIXIR query, the size of the intermediate data generated by WHIRL, and the number of similarity-join predicates.

Our implementation of ELIXIR extends XML-QL, but we are currently exploring ways of extending the ELIXIR algorithm to other XML query languages such as Quilt [1]. Another area of future research is query optimization. The ELIXIR query processor follows a strict three-stage policy for rewriting the original query, but this may be sub-optimal in some cases. We are exploring techniques for automatically searching the space of query rewritings in order to find one that is optimal with respect to standard metrics such as intermediate data sizes.

References

[1] D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML query language for heterogeneous data sources. In *Proc. SIGMOD/PODS Workshop on the Web and Databases*, 2000.

[2] W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proc. SIGMOD*, pages 201–211, 1998.

[3] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, , and D. Suciu. XML-QL: A query language for XML. In *Proc. 8th Int. World Wide Web Conf.*, 1999.

[4] A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with STORED. In *Proc. SIGMOD*, 1999.

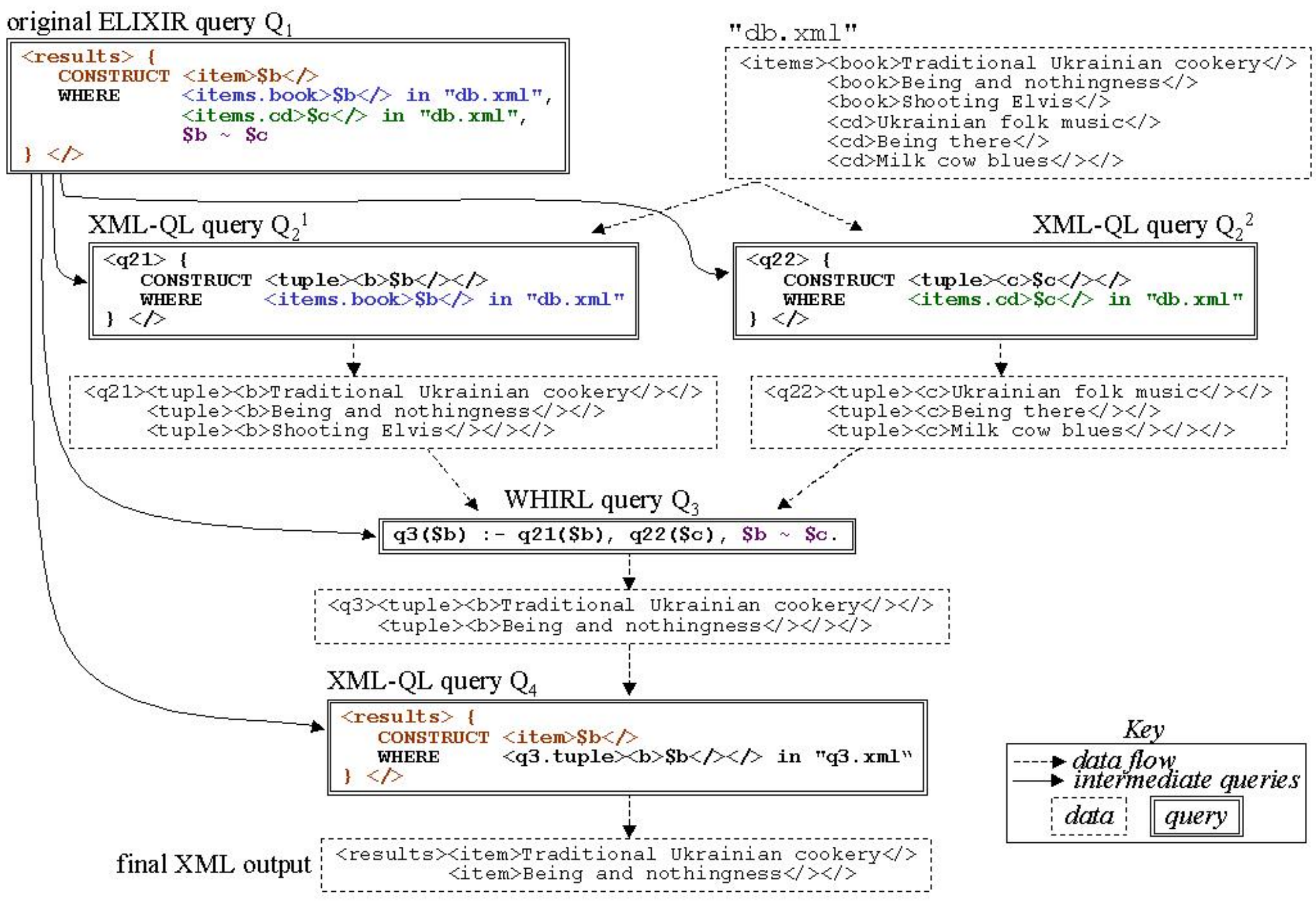
[5] D. Florescu, I. Manolescu, and D. Kossmann. Integrating keyword search into XML query processing. In *Proc. 9th Int. World Wide Web Conf.*, 2000.

[6] N. Fuhr and K. Großjohann. XIRQL: An extension of XQL for information retrieval. In *SIGIR Workshop on XML and Information Retrieval*, 2000.

[7] I. Macherius, G. Huck, and P. Fankhasuer. XQL extensions in the GMD-IPSI XQL Engine, 1999. http://xml.darmstadt.gmd.de/xql/extensions.

[8] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *Proc. Int. Conf. Very Large Databases*, 1999.

[9] A. Theobald and G. Weikum. Adding relevant to XML. In *Proc. SIGMOD/PODS Workshop on the Web and Databases*, 2000.



```
<results> {
   CONSTRUCT <item>$b</>
   WHERE     <items.book>$b</> in "db.xml",
             <items.cd>$c</> in "db.xml",
             $b ~ $c
} </>
```

```
<items><book>Traditional Ukrainian cookery</>
       <book>Being and nothingness</>
       <book>Shooting Elvis</>
       <cd>Ukrainian folk music</>
       <cd>Being there</>
       <cd>Milk cow blues</></>
```

```
<q21> {
   CONSTRUCT <tuple><b>$b</></>
   WHERE     <items.book>$b</> in "db.xml"
} </>
```

```
<q22> {
   CONSTRUCT <tuple><c>$c</></>
   WHERE     <items.cd>$c</> in "db.xml"
} </>
```

```
<q21><tuple><b>Traditional Ukrainian cookery</></>
     <tuple><b>Being and nothingness</></>
     <tuple><b>Shooting Elvis</></></>
```

```
<q22><tuple><c>Ukrainian folk music</></>
     <tuple><c>Being there</></>
     <tuple><c>Milk cow blues</></></>
```

```
q3($b) :- q21($b), q22($c), $b ~ $c.
```

```
<q3><tuple><b>Traditional Ukrainian cookery</></>
    <tuple><b>Being and nothingness</></></>
```

```
<results> {
   CONSTRUCT <item>$b</>
   WHERE     <q3.tuple><b>$b</></> in "q3.xml"
} </>
```

```
<results><item>Traditional Ukrainian cookery</>
         <item>Being and nothingness</></>
```

Figure 1: The ELIXIR query processor efficiently generates an answer to a query Q_1 ("find books and CDs with similar titles") by rewriting Q_1 into a series of XML-QL queries Q_2^i, a WHIRL query Q_3, and a final XML-QL query Q_4. Color indicates which parts of Q_B are used to generate the intermediate queries.

(a)

```
<results> {
    CONSTRUCT <similar><t>$t</><v>$v</><l>$l</></>
    WHERE     <SigmodRecord.issue><volume>$vol</><articles.article.title>$t</></> in "SR.xml",
              <tsts.bookcoll.book.chapter.v>$v</> in "NT.xml",
              <PLAY.ACT.SCENE.SPEEC.LINE>$l</> in "MB.xml",
              $vol > 24, $t ∼ $v, $t ∼ $l
} </>
```

(b)

```
<results>
    <similar><t>Opportunities in Information Management and Assurance.</>
             <v>And from that time he sought opportunity to betray him.</>
             <l>But yet I'll make assurance double sure,</></I>
    <similar><t>Size Separation Spatial Join.</>
             <v>But he that is joined unto the Lord is one spirit.</>
             <l>To Ireland, I; our separated fortune</></>
    <similar><t>Where Will Object Technology Drive Data Administration?</>
             <v>And there are differences of administrations, but the same Lord.</>
             <l>Where?</></>
                 ⋮
</>
```

Figure 2: An ELIXIR query to retrieve recent SIGMOD Record articles, verses from the New Testament, and lines from Macbeth that are all similar (a), and some of the answers discovered by ELIXIR (b).