The XML Query Language Xcerpt

Pattern Queries for XML and Semistructured Data

François Bry and Sebastian Schaffert

http://www.pms.informatik.uni-muenchen.de

Institut für Informatik, LMU München
Introduction

State-of-the-art for XML Query languages (XQuery, XSLT):

- path-oriented
  (e.g. /descendant::a/descendant::b[following-sibling::c])
- patterns serve to assemble path-selections

⇒ strong intertwining of query and construction parts
⇒ queries tend to be complicated
This XQuery expression creates a list of publications (titles) for each author.
Introduction – Xcerpt

- declarative, rule-based transformations and queries
- patterns for constructing results
- patterns for queries (instead of paths)
- strict separation of query and construction parts – “connected” via variables
- chaining rules instead of nesting queries
Contents

• Introduction
• Database, Query and Construct Terms
• Construct-Query Rules
• Xcerpt Semantics
• Comparison with XQuery
• Summary
Xcerpt programs build upon the following constructs:

- Database Terms represent semistructured databases / data items
- Query Terms are patterns for querying data
- Construct Terms are patterns for the result of a query

Construct-Query Rules relate Query Terms with Construct Terms.
Database Terms

- Database Terms are representations of XML documents
- Terms are of the form $l\{t_1, \ldots, t_n\}$ or $l[t_1, \ldots, t_n]$ where
  - $l$ is a label (i.e. tag name in XML)
  - $\{t_1, \ldots, t_n\}$ is an unordered (multi-)set of subterms
  - $[t_1, \ldots, t_n]$ is an ordered sequence of subterms
- Terms may contain references (making cycles possible)
  - $a : t$ associates the name $a$ with the subterm $t$
  - $\uparrow a$ references ("points to") the subterm named $a$
Database Terms – Example 1

bib {
    a1: author { last{ "Stevens" }, first { "W." } },
    a2: author { last{ "Abiteboul" }, first { "Serge" } },
    a3: author { last{ "Buneman" }, first { "Peter" } },
    a4: author { last{ "Suciu" }, first { "Dan" } },

    book {
        title { "TCP/IP Illustrated" },
        authors [ a1 ],
        publisher { "Addison-Wesley" },
        price { "65.95" }
    },
    book {
        title { "Advanced Programming in the Unix environment" },
        authors [ a1 ],
        publisher { "Addison-Wesley" },
        price { "65.95" }
    },
    book {
        title { "Data on the Web" },
        authors [ a2, a3, a4 ],
        publisher { "Morgan Kaufmann Publishers" },
        price { "39.95" }
    }
}

<reviews>
  <entry>
    <title>Data on the Web</title>
    <price>34.95</price>
    <review>
      A good discussion ...
    </review>
  </entry>
  <entry>
    <title>Advanced Programming in the Unix ...</title>
    <price>65.95</price>
    <review>
      A clear and detailed ...
    </review>
  </entry>
  <entry>
    <title>TCP/IP Illustrated</title>
    <price>65.95</price>
    <review>
      One of the best books ...
    </review>
  </entry>
</reviews>
Query Terms

Query Terms . . .

- specify (possibly incomplete) patterns for the data
- are somewhat similar to SQL selections and to goals in logic programming
Query Terms

Query Terms . . .

• specify (possibly incomplete) patterns for the data
• are somewhat similar to SQL selections and to goals in logic programming

. . . but have the following additional properties:

• Answers might have additional subterms to those in the query term.
• Answers might have a different subterm ordering than the query.
• A query term might specify subterms at an unspecified depth.
Query Terms

Query Terms may contain the following constructs:

- single curly or square brackets (\{\}, [\[]) denote a total term specification
- double curly or square brackets (\{{{\}}}, [[[]]]) denote a partial term specification
- variables (possibly with pattern restriction using ~\rightleftarrows – read “as”)
- the desc construct to specify subterm patterns at unspecified depth
Query Terms – Example 1

partial vs. total matching:

```
bib {
  book {
    title { "Data on the Web" },
  }
}
```

matches with the database

```
bib {
  book {
    title { "Data on the Web" },
  }
}  
```

```
bib {
  book {
    title { "TCP/IP Illustrated" },
    authors [ author { last { "Stevens" }, first { "W." } } ],
    publisher { "Addison-Wesley" },
    price { "65.95" }
  },
  book {
    title { "Data on the Web" },
    authors [ author { last { "Abiteboul" }, first { "Serge" } },
                author { last { "Buneman" }, first { "Peter" } },
                author { last { "Suciu" }, first { "Dan" } } ],
    publisher { "Morgan Kaufmann Publishers" },
    price { "39.95" }
  },
  ...
}  
```

does not match with the database
unrestricted and restricted variables:

\[
\text{bib} \{ \\
\quad \text{book} \{ \\
\quad \quad \text{X,} \\
\quad \quad \quad \text{authors} \{ \{ \text{AUTHOR} \} \} \\
\quad \} \\
\}\]

binds e.g. to

\[
\text{X} = \text{price}\{ "39.95" \} \text{ AND} \\
( \text{AUTHOR} = \text{author}\{ \\
\quad \text{first}\{ "Dan" \}, \\
\quad \text{last}\{ "Suciu" \} \} \text{ OR} \\
\text{AUTHOR} = \text{author}\{ \\
\quad \text{first}\{ "Peter" \}, \\
\quad \text{last}\{ "Buneman" \} \} \text{ OR} \\
\text{AUTHOR} = \text{author}\{ \\
\quad \text{first}\{ "Serge" \}, \\
\quad \text{last}\{ "Abiteboul" \} \} )
\]

\[
\text{bib} \{ \\
\quad \text{book} \{ \\
\quad \quad \text{Y} \text{ \Leftrightarrow title,} \\
\quad \quad \quad \text{authors} \{ \{ \text{AUTHOR} \} \} \\
\quad \} \\
\}\]

binds e.g. to

\[
\text{Y} = \text{title}\{ "Data on the Web" \} \text{ AND} \\
( \text{AUTHOR} = \text{author}\{ \\
\quad \text{first}\{ "Dan" \}, \\
\quad \text{last}\{ "Suciu" \} \} \text{ OR} \\
\text{AUTHOR} = \text{author}\{ \\
\quad \text{first}\{ "Peter" \}, \\
\quad \text{last}\{ "Buneman" \} \} \text{ OR} \\
\text{AUTHOR} = \text{author}\{ \\
\quad \text{first}\{ "Serge" \}, \\
\quad \text{last}\{ "Abiteboul" \} \} )
\]
the \textit{descendant} construct:

\begin{verbatim}
bib {
    book {
        TITLE \leadsto title,
        authors {{ desc "Stevens" }}
    }
}
\end{verbatim}

matches against a database term with "Stevens" at any depth below the \texttt{authors} element.
A non-standard unification is necessary:

- **partial patterns** (i.e. `{ { } }` and `[ [ ] ]`) have to be treated properly (i.e. the database may contain additional subterms not specified in the query),

- the **descendant** construct which is usually not treated in other unification methods, must be dealt with,

- **ordered and unordered** terms (`{ }` and `[ ]`) have to be handled according to their semantics.
A *Simulation* is a relation between two graphs where one graph can be represented in the other (wrt labels and edges):

Term $t_1$ simulates into term $t_2$ – written $t_1 \preceq t_2$
Comparison with XQuery – XQuery

<results>
{ 
  for $b in 
  return 
    <result>
      { $b/title } 
      { $b/author  }
    </result>
}
</results>

<results>
{ 
  for $a in 
    distinct-values(document("http://bn.com")//author)
  return 
    <result>
      { $a }
      { 
          where some $ba in $b/author
          satisfies deep-equal($ba,$a)
        return $b/title
      }
    </result>
}
</results>

All authors for each title

All titles for each author
Comparison with XQuery – Xcerpt

rule { cons {
  results {
    all result {
      TITLE,
      all AUTHOR
    }
  }
},
query {
  in { "http://bn.com" },
bib {{
    book {{
      TITLE ~> title,
      authors {{
        AUTHOR ~> author
      }}
    }}
  }}
}
}

All authors for each title

All titles for each author
Summary

Main features of Xcerpt, a query and transformation language for XML:

- declarative, rule- and term-based querying
- Database Terms represent semistructured databases
- Query Terms are templates that are unified with a database (or another rule)
- Construct Terms are templates for the result, replacing variables with their bindings
- multiple rules may be chained together
- deductive language for both querying/transforming data and deductive reasoning
Thank you!

Xcerpt is an ongoing research project. For more information, please visit http://www.xcerpt.org