Foundations of XML Data Manipulation

Giorgio Ghelli

Query Languages for SSD and XML

Some names

- UnQL: BunDavHil96
- Lorel: AbiQuaMcH97
- XSLT, XPath, XQuery: google:w3c xslt, w3c xpath, w3c xquery
- XML-QL: XMLQL, xquery99
- XDuce: HosPie03
- TQL: CarGhe03
- YATL, Strudel…

Path expressions

<bib>
<book year="**1995**">
<author> <first>**Serge**</first> <last>**Abiteboul**</last> </author>
<author> <first>**Richard**</first> <last>**Hull**</last> </author>
<author> <first>**Victor**</first> <last>**Vianu**</last> </author>
<publisher>**Addison**</publisher>
<price>**60**</price>
</book>
<book year="**1993**">
<title>**Formal Semantics**</title>
<author><first>**Glynn**</first> <last>**Winskel**</last> </author>
<publisher>**MIT Press**</publisher>
<price>**42**</price>
</book>
</bib>

Path expressions

- *Document* /bib/book/author/first:
  - <first>Serge</first>, <first>Richard</first>, <first>Victor</first>, <first>Glynn</first>
- Semantics:
  - All nodes you find starting from *Document* and walking down a /bib/book/author/first path
  - More generally: $[[p]] = \{ \langle n,m \rangle \mid m \in n/p \}$
- The interesting case:
  - Data is a graph
  - Paths is a regular expression

A formal definition

- Assume a graph $G=(N,E)$ with $E \subseteq N \times A \times N$
- A word w in A* determines a relation |w| on $N \times N$:
  - $n\ |\varepsilon|\ n$
  - $(n,a,n') \in E \Rightarrow n\ |a|\ n'$
  - $n\ |w|\ n'$ and $n'\ |w'|\ n'' \Rightarrow n\ |w.w'|\ n''$
- A language $L \subseteq A^*$ determines a relation:
  - $n\ |L|\ n' \Leftrightarrow n\ |w|\ n'$ for some $w \in L$
- A regexp r determines a relation:
  - $n\ |r|\ n' \Leftrightarrow n\ |Lang(r)|\ n'$
- We allow regexp on labels as well

Regular path expressions

- "book"."[T|t]itle":
  – {book.title, book.Title}
- book.(Title|title):
  – {book.title, book.Title}
- ".*".title:
  – {book.title, author.title, name.title,…} i.e. {_.title}
- book.refs*.title:
  – {book.title, book.refs.title, book.refs.refs.title,…}
- (".*")*.name:
  – {name, _.name, _._.name, _._._.name,…}

Syntax for path expressions

- a ::= ε | letter | . | a a | [a|a] | a*
- r ::= ε | a | _ | r.r | r|r | e*
- Abbreviations:
  – r? = r|ε
  – r+ = r.r*

XPath syntax

- p,p' ::= / p | p/p' | axis::nodetest
- nodetest ::= * | tag | text() | element(*) | element(tag) | attribute(*) | …
- axis ::= child | parent | d-o-s | a-o-s
  | descendant | ancestor
  | following | preceding
  | following-sibling | preceding-sibling
- p/nodetest = p/child::nodetest
- p//q = p/d-o-s::*/q
- .. = parent::node()

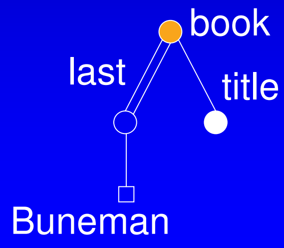
Tree Patterns

- Titles of books:

  $doc //book/title

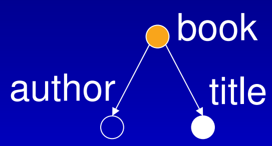
- Titles of books by Buneman:

  $doc //book[//last/text() = 'Buneman']/title

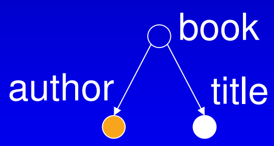


Tree Patterns (Twigs)

- book[author]/title
- book/author/../title



- author/../book/title



- Up pointers are slightly more expressive than twigs

Are path expressions a query language?

The structure of XML query languages

- FROM BindingExpression(X): generate a set of bindings for X
- WHERE Condition(X): filter some bindings out
- SELECT Result(X): evaluate Result(X) once for each binding, and find a way to merge the results
- Path expressions come handy in the FROM clause

MicroXQuery

- Reference: Colazzo et al., Types for Path Correctness of XML Queries, ICALP'04.

```
for  $b in $doc /bib/book,
     $a in $b /author
where $b /@year > 2000
return <libro> $b/title, $a </libro>
```

MicroXQuery

- for $b in $doc /bib/book

  let $a = $b /author

  where $b /@year > 2000

  return <libro> $b/title, $a </libro>

- for $b in $doc /bib/book

  where $b /@year > 2000

  return <libro> $b/title, $b /author </libro>

Tree Patterns

- Titles of books by Buneman:

  for $x in $doc //book

  where $x/authors/last/text() = 'Buneman'

  return $x/title

- Same as:

  $doc //book[last/text() = 'Buneman']/title

Trees to relations to trees

```
let  $authors = $doc /bib/book/author
for $a in distinct($authors)
return <booksByAuth> $a,
            for  $bb in $doc /bib/book
            where  $a isin $bb /author
            return $bb/title
    </booksByAuth>
```

Result

- From
  - <bib>
  - (<book> (<author></author>)*
  - </book>)*
  - </bib>
- To
  - (<booksByAuth>
  - <author>…</author>
  - (<title>…</title>)*
  - </booksByAuth>)*

Yet another syntax

```
BBAS[
let  $authors = $doc /bib/book/author
for $a in distinct($authors)
return booksByAuth[
          $a,
          for  $bb in $doc /bib/book
          where  $a isin $bb /author
          return $bb/title
       ]
]
```

Type

- Type:

  BBAS[booksbyaut[author[String], title[String]*]*]

- If:

```
$doc: bib[  book [   title[String],
                     author[String]*
            ]*
       ]
```

Tree manipulation

- Micro XQuery is able to do “nested relations”, i.e. trees with fixed depth
- It is unable to produce arbitrarily deeply nested trees:
  - Structural recursion
- It is unable to reverse a graph:
  - Skolem functions

Structural recursion

- f(v) => v
- f(author[x]) => autore[f(x)]
- f(l[x]) => if l = year then 0 else l[f(x)]
- f(0) => 0
- f(x,y) => f(x),f(y)

Structural recursion

- f: collect authors, if any
- f(author[x],y) => authors[author[f(x)], g(y)], h(y)
- g(author[x]) => author[f(x)]
- (else) g(l[x]) => 0
- h(author[x]) => 0
- (else) h(l[x]) => l[f(x)]
- Implicit: f(a[x],y) => f(a[x]),f(y); f(a[x]) => a[f(x)]

Structural recursion vs. query languages

- Paths go down
- for – where – selects iterates horizontally
- Structural recursion does both
- Supported by:
  - XDuce (SR only)
  - XSL (SR only)
  - XQuery (FWR + recursion)

XSL

```
<xsl:stylesheet version="1.0" ...>
   <xsl:template match="ul[parent::ul]">
      <li>
            <ul>
                  <xsl:apply-templates select="@*|node()"/>
            </ul>
      </li>
   </xsl:template>
   <xsl:template match="@*|node()">
      <xsl:copy>
            <xsl:apply-templates select="@*|node()"/>
      </xsl:copy>
   </xsl:template>
</xsl:stylesheet>
```

Skolem functions

- new(a,b,c,…): a (new) node
- n-lab-n: an edge

- for $x-$lab-$y in Edges
- return new(node,$y)-$lab-new(node,$x)

- for $x-$lab-$y in Edges
- return new(node,$x)-$lab-new(node,$y),
- new(node,$y)-home-new(root)

Logical languages: TQL

– Matching through ambient-logic formulas:

```
FROM  $db |= .paper[ (.author[$X] or .autore[$X])
                      and not .editor[$X]
                   ]
SELECT author[$X]
```

Returns:

author[Cardelli] | author[Gordon] | author[Ghelli]

Find All Keys

```
from $Bib
  |= bib[!book[.$k[T]]
        And foreach $X.
          Not (.book.$k[$X] | .book.$k[$X])
      ]
select key[$k]
```

The Logic

- $F \vdash_\sigma 0$ iff $F=0$
- $F \vdash_\sigma A \mid B$ iff $\exists F', F''.\ F = F'|F'', F' \vdash_\sigma A, F'' \vdash_\sigma B$
- $F \vdash_\sigma \eta[A]$ iff $F= \sigma(\eta)[F'], F' \vdash_\sigma A$
- $F \vdash_\sigma A \wedge B$ iff $F \vdash_\sigma A$ and $F \vdash_\sigma B$
- $F \vdash_\sigma \neg A$ iff not $(F \vdash_\sigma A)$
- $F \vdash_\sigma \eta=\eta'$ iff $\sigma(\eta)=\sigma(\eta')$
- $F \vdash_\sigma \exists x.A$ iff $\exists n.\ F \vdash_{\sigma\{n/x\}} A$
- $F \vdash_\sigma \exists X.A$ iff $\exists F'.\ F \vdash_{\sigma\{F'/X\}} A$
- $F \vdash_\sigma \mu\xi.A$ iff $F \vdash_\sigma A\{\mu\xi.A/\xi\}$ (is circular…)

- De Morgan duals: $||$, $\eta[\Rightarrow A]$, F, $\vee$, $\neq$, $\forall$, $\nu\xi.A$, ! $\eta[A]$

Logical languages: monadic datalog

- doc //book/author X:
  - result(Y) :- desc(doc,X), name(X,book), child(X,Y), name(Y,author)

The full hybrid mu-calculus

- A ::= i | lab | <s>A | A ∧ A | ¬A | μξ.A | ξ
- E,L,w $\vdash_\sigma$ i iff L(i) = {w}
- E,L,w $\vdash_\sigma$ lab iff w∈L(lab) (L(w) = lab)
- E,L,w $\vdash_\sigma$ <s>A iff ∃w’. w.s.w’ in E and E,L,w’ $\vdash_\sigma$ A
- E,L,w $\vdash_\sigma$ <-s>A iff ∃w’. w’.s.w in E and E,L,w’ $\vdash_\sigma$ A
- E,L,w $\vdash_\sigma$ A ∧ B iff E,L,w $\vdash_\sigma$ A and E,L,w $\vdash_\sigma$ B
- E,L,w $\vdash_\sigma$ ¬A iff not (E,L,w $\vdash_\sigma$ A)
- E,L,w $\vdash_\sigma$ μξ.A iff E,L,w $\vdash_\sigma$ A{μξ.A/ξ} (is circular…)
- Over XML:
  - Just two steps s: ↓/↑ (firstchild) and →/← (nextsibling)
  - E is a finite tree
  - L(lab) is a partition of the nodes
- De Morgan duals: [s]A, ∨, νξ.A

Encoding axes

- <child> A = <↓>(μξ (A ∨ <→>ξ))
- <parent> A = μξ (<↑>A ∨ <←>ξ)
- <desc> A = <child>(μξ (A ∨ <child>ξ))
- <ancestor>
- <following-sibling>
- [everywhere] A = νξ (A ∧ [↓]ξ ∧ [→]ξ)
- <somewhere> A = μξ (A ∨ <↓>ξ ∨ <→>ξ)

Logical languages: modal mu calculus

- <m,n> in [[p]] iff E,L,i->{n}, m |= <<p>>
- //book/author:
  <desc>(book /\ <child>(author /\ i))
- <m,n> in //book[title]/author
  i->n, m |=
  <desc>(book and (<child>title)
  and (<child>author and i))
- <m,n> in //book[not title]/author
  i->n, m |=
  <desc>(book and (not <child>title)
  and (<child>author and i))

Logical languages: MSO

- MTran: transformation language based on MSO [HinabaHosoya..]
- MSO: FO plus set quantification
- Able to express all regular tree query

MTran formulas

- //img: x in <img>
- //*[date]: ex1 y: x/y and y in <date>
- //a{x}/b{y}: x in <a> and x/y and y in <b>

MTran transformation

- Add an <li> around to any <ul> whose parent is an <ul>:
  - { visit x :: /x & x in <ul> :: li[x] }
- *visit* copies all unmatched nodes, *gather* does not:
  - {gather x :: x in <a> :: li[x]}

Nesting

- {gather b :: b in <book> ::
  {gather a :: b/a & a in <author> ::
  book-author[b,a] } }

Evaluations

- Compile the formula to a tree automaton
  - Non-elementary in the worth case
  - MONA often works well
- Evaluate the automaton
  - Linear algorithm (non trivial, since the query is n-ary)

Tree Automata

- Morally, the same expressive power (over trees) as:
  - Monadic datalog
  - Tree patterns
  - MSO
  - Mu-calculus
  - XDuce type system
  - …
- Things are not so simple…

Readings

- Xquery99: comparison of XML-QL, YATL, Lorel, XQL
- BonCer00: comparison of Lorel, XML-QL, XML-GL, XSL, XQL
- ColGheAl06: MicroXQuery
- www.w3.org/TR/xquery|xpath|xslr: XQuery, XPath, XSLT
- BunDavHil96: UnQL
- AbiQuaMcH97: Lorel
- HosPie03: XDuce
- klarlundSchweintick: XPath, XQuery, XSLT
- CarGhe03: TQL
- Tocl-lics0203: Logics, Automata