A Functional Data Model and Algebra for XML Query

LiComR Summer Workshop 2003
2003. 8. 21

배 민 오

동덕여자대학교

Outline

- XML(eXtensible Markup Language)
  - DTD
- XML Query
  - XML-QL
  - Yatl
- Data Model
  - Nested Relational Model
- Algebra
  - Haskell
- Examples
- Conclusions

XML Serialization


```
<bib>
<book year="1999">
    <title>Data on the Web</title>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
    <publisher>Addison-Wesley</publisher>
</book>
<book year="1987">
    <title>Foundations of databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
</book>
</bib>
```

- element→nested block structure
- dom tree

DTD(Document Type Def)

```
<!ELEMENT bib (book*)>
<!ELEMENT book (title, author+, publisher)>
<!ATTLIST book year CDATA #REQUIRED >
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
```

속성의 타입을 정의

Parsed Char Data

EBNF로 문서의 구조를 정의함

example in XML-QL

```
CONSTRUCT <bib> {
        WHERE
<bib>
    <book year=$y>
        <title>$t</title>
        <publisher>Addison-Wesley</publisher>
    </book>
</bib> IN "www.bn.com/bib.xml",
$y > 1991
CONSTRUCT <book year=$y><title>$t</title></book>
}</bib>
```

```
<bib>
<book year="1999">
    <title>Data on the Web</title>
</book>
</bib>
```


example in Yatl

```
make
bib [ *book [ @year [ $y ], title [ $t ] ] ]
match "www.bn.com/bib.xml" with
bib [ *book [ @year [ $y ], title [ $t ], publisher [ $p ] ]
where
$p = "Addison-Wesley" and $y > 1991
```

Data Model

- data model for XSLT[Wadler 1999]
- XSLT recommendation[W3C 1999]

- addition of reference nodes
- merge attribute and element nodes
- eliminate comment and PI nodes

Constructor functions

basic type: Node

1. text
2. element
3. reference

```
text  :: String -> Node
elem  :: Tag -> [Node] -> Node
ref   :: Node -> Node
```

List of Nodes

Model term

```
elem "bib" [                                              -->bib0
    elem "book" [                                         -->book0
        elem "@year" [ text "1999" ],                     -->year0
        elem "title" [ text "Data on the Web" ],
        elem "author" [ text "Abiteboul" ],
        elem "author" [ text "Buneman" ],
        elem "author" [ text "Suciu" ]],
    elem "book" [
        elem "@year" [ text "1987" ],
        elem "title" [ text "Foundations of Databases" ],
        elem "author" [ text "Abiteboul" ],
        elem "author" [ text "Hull" ],
        elem "author" [ text "Vianu" ]]]
```


type of Node

```
isText      :: Node -> Bool
isElem      :: Node -> Bool
isRef       :: Node -> Bool
text node           → access the text.
    string :: Node -> String
element node        → access tag, children.
    tag    :: Node -> Tag
    children :: Node -> [Node]
reference node      → access the node referenced.
    dereference :: Node -> Node
```

Attribute and Tag

check a tag to see whether it is an attribute (begins with @).

```
tagAttr :: Tag -> Bool
```

check whether a node is an element with a given tag, and whether it is an attribute.

```
is          :: Tag -> Node -> Bool
is t x      = isElem x  &&  tag x  == t
isAttr      :: Node -> Bool
isAttr x    = isElem x  && tagAttr (tag x)
```

is "@year" year0 =➔ true

isAttr year0 =➔ true

value: content of the node

1. text node=➔ its string
2. attribute =➔ value of the attribute
3. element node=➔ concatenation of the value of the non-attribute children

```
value :: Node -> String
```

value year0 =➔ "1999"

Types of Attribute

Attribute	multiplicity	child type
CDATA	one	text
NMTOKEN	one	text
NMTOKENS	many	text
ID	one	text
IDREF	one	reference
IDREFS	many	reference

Nested relational algebra

•relational approach to databases →table

•nested relational approach →

tuples and lists,

arbitrarily nested

Tuples

(1999,"Data on the Web",["Abiteboul","Buneman","Suciu"])

:: (Int,String,[String])

To decompose values, we allow tuples to appear on the left-hand side of a definition.

```
year :: (Int,String,[String])
year (x,y,l) = x
```


list comprehension

```
[ value x | x <- children book0, is "author" x ]
```

==> ["Abiteboul", "Buneman", "Suciu"]

```
[ value y | x <- children bib0, is "book" x,
            y <- children x, is "title" y ]
```

==> ["Data on the Web", "Foundations of Databases"]

comprehension

[exp | qual1, ..., qualn]

exp : return expression

$qual_i$: qualifier

filter bool-exp

generator pat <- list-exp

Using comprehensions to write queries

```
follow :: Tag -> Node -> [Node]
follow t x = [ y | y <- children x, is t y ]

[ value x | x <- follow "author" book0 ]
        ==> [ "Abiteboul", "Buneman", "Suciu" ]
[ value y | x <- follow "book" bib0, y <- follow "title" x ]
        ==> [ "Data on the Web", "Foundations of Databases" ]
```

compute cartesian products

```
[ (value y, value z)
        | x <- follow "book" bib0,
          y <- follow "title" x,
          z <- follow "author" x ]
   ==> [ ("Data on the Web", "Abiteboul"),
         ("Data on the Web", "Buneman"),
         ("Data on the Web", "Suciu"),
         ("Foundations of Databases", "Abiteboul"),
         ("Foundations of Databases", "Hull"),
         ("Foundations of Databases", "Vianu") ]
```


Comprehensions may be nested

```
[ (int (value y), value z, [ value u | u <- follow "author" x ])
    | x <- follow "book" bib0,
      y <- follow "@year" x,
      z <- follow "title" x ]
==>
[(1999, "Data on the Web", ["Abiteboul","Buneman","Suciu"]),
 (1991,"Foundations of Databases",["Abiteboul,"Hull","Vianu"])
]
```

book reviews

```
elem "reviews" [                                    --→ reviews0
    elem "book" [
        elem "title" [ text "Data on the Web" ],
        elem "review" [ text "This is great!" ]]
    elem "book" [
        elem "title" [ text "Foundations of Databases" ],
        elem "review" [ text "This is pretty good too!" ]]]
```


joins the two data sources

```
[ (value y, int (value z), value w)
    | x <- follow "book" bib0, y <- follow "title" x,
    z <- follow "@year" x, u <- follow "book" reviews0,
    v <- follow "title" u, w <- follow "review" u, y == v ]
==>
[("Data on the Web", 1999, "This is great!"),
("Foundations of Databases", 1991, "This is pretty good too!")]
```

Additional operations

```
(++) :: [a] -> [a] -> [a]

follow "title" book0 ++ follow "author" book0
==>
[elem "title" [text "Data on the Web"],
    elem "author" [ text "Abiteboul" ],
    elem "author" [ text "Buneman" ],
    elem "author" [ text "Suciu" ]]
```


index

```
index :: [a] -> [(Int,a)]

index (follow "author" book0)
==>
[(0, elem "author" [ text "Abiteboul" ]),
 (1, elem "author" [ text "Buneman" ]),
 (2, elem "author" [ text "Suciu" ])]

[ x | (i, x) <- index (follow "author" book0), i < 2 ]
==> [(0, elem "author" [ text "Abiteboul" ]),
       (1, elem "author" [ text "Buneman" ])]
```

empty & unique

```
null :: [a] -> Bool

null [ x | (i, x) <- index (follow "author" book0), i >= 1 ]
==> False
```

```
the :: [a] -> a

the (follow "title" book0)
==> elem "title" [text "Data on the Web"]
```


Structural recursion

```
value :: Node -> String
value x =
if isText x then
    string x
    else if isElem x then
        concat [ value y | y <- children x, not (isAttr y) ]
    else if isRef x then
        ""

concat :: [String] -> String
```

bookstore

```
<bookstore>
<fiction>
    <sci-fi>
        <book><isbn>0006482805</isbn>
        <title>Do androids dream of electric sheep</title>
        <author>Philip K. Dick</author>
        </book>
    </sci-fi>
    <fantasy><mystery>
        <book><isbn>0261102362</isbn>
        <title>The two towers</title><author>JRR Tolkien</author>
        </book>
    </mystery></fantasy>
</fiction>
</bookstore>
```


isbns

```
<bookstore>
<fiction>
<sci-fi><book><isbn>0006482805</isbn></book></sci-fi>
<fantasy><mystery><book><isbn>0261102362</isbn></book>
</mystery></fantasy>
</fiction>
</bookstore>
```

```
isbns :: Node -> Node
isbns x = if is "book" x then
    elem "book" [ the (follow "isbn" x) ]
else
    elem (tag x) [ isbns y | y <- children x ]
```

Regular expression matching

•Reg a ––stands for a regular expression that returns a value of type a for each successful match.

```
reg0=([ (x,y,u) | x <- anywhere (item "@year“),
          y <- item "title", u <- rep (item "author") ])
:: Reg (Node,Node,[Node])
```

```
match reg0 book0
==>
[(elem "@year" [text "1999"],
  elem "title" [text "Data on the Web"],
  [elem "author" [text "Abiteboul"],
   elem "author" [text "Buneman"],
   elem "author" [text "Suciu"]])]
```


match

```
match :: Reg a -> Node -> [a]

nodeItem :: Reg Node

textItem :: Reg Node
textItem = ([ x | x <- nodeItem, isText x ])

item :: Tag -> Reg Node
item t = ([ x | x <- nodeItem, is t x ])

alternation
(+++) :: Reg a -> Reg a -> Reg a
item "author" +++ item "editor"

([ int (value x) | x <- item "@year" ]) +++ ([ 1999 | True ])
```

repetition, string

```
repetition
rep :: Reg a -> Reg [a]
rep p = ([ [x]++ | x <- p, | <- rep p ]) +++ ([ [] | True ])

stringItem :: Reg String
stringItem = ([ string x | x <- textItem ])
```


step

```
step :: Tag -> Reg a -> Reg a

step "@year" stringItem :: Reg String

step "name" ([ (x,y) | x <- step "first" stringItem,
        y <- step "last“ stringItem ]) :: Reg (String, String)

([ (int x,y,u) | x <- anywhere (step "@year" stringItem),
                 y <- step "title" stringItem,
                 u <- rep (step "author" stringItem) ])
:: Reg (Int,String,[String])
```

Query for XML-QL

```
query :: Node -> Node
query x = elem "bib"
[ elem "book" [ elem "@year" [text (value y)], elem "title" [text (value t)]]
| a <- follow "bib" x, b <- follow "book" a,
y <- follow "@year" b, t <- follow "title" b,
p <- follow "publisher" b,
int (value y) > 1991, value p == "Addison-Wesley" ]
```

```
elem "bib" [
    elem "book" [
        elem "@year" [ text "1999" ],
        elem "title" [ text "Data on the Web" ]]]
```


Query for Yatl

```
query :: Node -> Node
query x = elem "bib"
[ elem "book"
[ elem "@year" [ text y ], elem "title" [ text t ] ]
        | (y,t,n) <- the (match bibReg x), int y > 1991,
        p == "Addison Wesley" ]
bibReg :: Reg [(String,String,String)]
bibReg = step "bib"
(rep (step "book" ([ (y,t,p) | y <- step "@year" stringItem,
                        t <- step "title" stringItem,
                        p <- step "publisher" stringItem ])))
```

Conclusions

- nested relational algebra
  - widely used for semistructured and OODBs
- list comprehension
- regular expressions
  - DTD, Schema
- function PL→Haskell
- algebra→logical(vs. physical) level

FUNDAMENTAL Relational operators

- selection $\sigma_{condition}(R)$
- projection $\pi_{att-list}(R)$
- cartesian product MALE x FEMALE
- set union R ∪ S
- set difference R − S

$$r \div s = \pi_{(R-S)}(r) - \pi_{(R-S)}[(\pi_{(R-S)}(r) \times s) - r]$$