XQuery, A typed functional language for querying XML

Philip Wadler, Avaya Labs

`wadler@avaya.com`

The Evolution of Language

$2x$ (Descartes)

$\lambda x.\,2x$ (Church)

`(LAMBDA (X) (* 2 X))` (McCarthy)

```
<?xml version="1.0"?>
<LAMBDA-TERM>
  <VAR-LIST>
    <VAR>X</VAR>
  </VAR-LIST>
  <EXPR>
    <APPLICATION>
      <EXPR><CONST>*</CONST></EXPR>
      <ARGUMENT-LIST>
        <EXPR><CONST>2</CONST></EXPR>
        <EXPR><VAR>X</VAR></EXPR>
      </ARGUMENT-LIST>
    </APPLICATION>
  </EXPR>
</LAMBDA-TERM>
```

(W3C)

Acknowledgements

This tutorial is joint work with:

Mary Fernandez (AT&T)

Jerome Simeon (Lucent)

The W3C XML Query Working Group

Disclaimer: This tutorial touches on open issues of XQuery. Other members of the XML Query WG may disagree with our view.

"Where a mathematical reasoning can be had, it's as great folly to make use of any other, as to grope for a thing in the dark, when you have a candle standing by you."

— Arbuthnot

Part I

XQuery by example

XQuery by example

Titles of all books published before 2000

```
/BOOKS/BOOK[@YEAR < 2000]/TITLE
```

Year and title of all books published before 2000

```
for $book in /BOOKS/BOOK
where $book/@YEAR < 2000
return <BOOK>{ $book/@YEAR, $book/TITLE }</BOOK>
```

Books grouped by author

```
for $author in distinct(/BOOKS/BOOK/AUTHOR) return
  <AUTHOR NAME="{ $author }">{
    /BOOKS/BOOK[AUTHOR = $author]/TITLE
  }</AUTHOR>
```

Part I.1

XQuery data model

Some XML data

```
<BOOKS>
  <BOOK YEAR="1999 2003">
    <AUTHOR>Abiteboul</AUTHOR>
    <AUTHOR>Buneman</AUTHOR>
    <AUTHOR>Suciu</AUTHOR>
    <TITLE>Data on the Web</TITLE>
    <REVIEW>A <EM>fine</EM> book.</REVIEW>
  </BOOK>
  <BOOK YEAR="2002">
    <AUTHOR>Buneman</AUTHOR>
    <TITLE>XML in Scotland</TITLE>
    <REVIEW><EM>The <EM>best</EM> ever!</EM></REVIEW>
  </BOOK>
</BOOKS>
```

Data model

XML

```
<BOOK YEAR="1999 2003">
  <AUTHOR>Abiteboul</AUTHOR>
  <AUTHOR>Buneman</AUTHOR>
  <AUTHOR>Suciu</AUTHOR>
  <TITLE>Data on the Web</TITLE>
  <REVIEW>A <EM>fine</EM> book.</REVIEW>
</BOOK>
```

XQuery

```
element BOOK {
  attribute YEAR { 1999, 2003 },
  element AUTHOR { "Abiteboul" },
  element AUTHOR { "Buneman" },
  element AUTHOR { "Suciu" },
  element TITLE { "Data on the Web" },
  element REVIEW { "A ", element EM { "fine" },  " book." }
}
```

Part I.2

XQuery types

DTD (Document Type Definition)

```
<!ELEMENT BOOKS (BOOK*)>
<!ELEMENT BOOK (AUTHOR+, TITLE, REVIEW?)>
<!ATTLIST BOOK YEAR CDATA #OPTIONAL>
<!ELEMENT AUTHOR (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ENTITY % INLINE "( #PCDATA | EM | BOLD )*">
<!ELEMENT REVIEW %INLINE;>
<!ELEMENT EM %INLINE;>
<!ELEMENT BOLD %INLINE;>
```

Schema

```
<xsd:schema targetns="http://www.example.com/books"
            xmlns="http://www.example.com/books"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            attributeFormDefault="qualified"
            elementFormDefault="qualified">
  <xsd:element name="BOOKS">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="BOOK"
          minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

Schema, continued

```
<xsd:element name="BOOK">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="AUTHOR" type="xsd:string"
         minOccurs="1" maxOccurs="unbounded"/>
      <xsd:element name="TITLE" type="xsd:string"/>
      <xsd:element name="REVIEW" type="INLINE"
         minOccurs="0" maxOccurs="1"/>
    <xsd:sequence>
    <xsd:attribute name="YEAR" type="INTEGER-LIST"
      use="optional"/>
  </xsd:complexType>
</xsd:element>
```

Schema, continued[2]

```
  <xsd:complexType name="INLINE" mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="EM" type="INLINE"/>
      <xsd:element name="BOLD" type="INLINE"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:simpleType name="INTEGER-LIST">
    <xsd:list itemType="xsd:integer"/>
  </xsd:simpleType>
</xsd:schema>
```

XQuery types

```
define element BOOKS of type BOOKS-TYPE
define type BOOKS-TYPE {
  element BOOK of type BOOK-TYPE *
}
define type BOOK-TYPE {
  attribute YEAR of type INTEGER-LIST ? ,
  element AUTHOR of type xs:string + ,
  element TITLE of type xs:string ,
  element REVIEW of type INLINE ?
}
define type INLINE mixed {
  ( element EM of type INLINE |
    element BOLD of type INLINE ) *
}
define type INTEGER-LIST {
  xs:integer *
}
```

Data model with types

XQuery

```
element BOOK of type BOOK-TYPE {
  attribute YEAR of type INTEGER-LIST { 1999, 2003 },
  element AUTHOR of type xs:string { "Abiteboul" },
  element AUTHOR of type xs:string { "Buneman" },
  element AUTHOR of type xs:string { "Suciu" },
  element TITLE of type xs:string { "Data on the Web" },
  element REVIEW of type INLINE {
    "A ",
    element EM of type INLINE { "fine" },
    " book."
  }
}
```

Part I.3

XQuery and Schema

XQuery and Schema

Authors and title of books published before 2000

```
schema "http://www.example.com/books"
namespace default = "http://www.example.com/books"
validate
  <BOOKS>{
    for $book in /BOOKS/BOOK[@YEAR < 2000] return
      <BOOK>{ $book/AUTHOR, $book/TITLE }</BOOK>
  }</BOOKS>
```

$\in$

```
element BOOKS {
  element BOOK {
    element AUTHOR { xsd:string } +,
    element TITLE { xsd:string }
  } *
}
```

Another Schema

```
<xsd:schema targetns="http://www.example.com/answer"
            xmlns="http://www.example.com/answer"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            elementFormDefault="qualified">
  <xsd:element name="ANSWER">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="BOOK"
          minOccurs="0" maxOccurs="unbounded"/>
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="TITLE" type="xsd:string"/>
              <xsd:element name="AUTHOR" type="xsd:string"
                 minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Another XQuery type

```
element ANSWER { BOOK* }
element BOOK { TITLE, AUTHOR+ }
element AUTHOR { xsd:string }
element TITLE { xsd:string }
```

XQuery with multiple Schemas

Title and authors of books published before 2000

```
schema "http://www.example.com/books"
schema "http://www.example.com/answer"
namespace B = "http://www.example.com/books"
namespace A = "http://www.example.com/answer"
validate
  <A:ANSWER>{
    for $book in /B:BOOKS/B:BOOK[@YEAR < 2000] return
      <A:BOOK>{
        <A:TITLE>{ $book/B:TITLE/text() }</A:TITLE>,
        for $author in $book/B:AUTHOR return
          <A:AUTHOR>{ $author/text() }</A:AUTHOR>
      }<A:BOOK>
  }</A:ANSWER>
```

Part I.4

Projection

Projection

Return all authors of all books

```
/BOOKS/BOOK/AUTHOR
```

$\Rightarrow$

```
<AUTHOR>Abiteboul</AUTHOR>,
<AUTHOR>Buneman</AUTHOR>,
<AUTHOR>Suciu</AUTHOR>,
<AUTHOR>Buneman</AUTHOR>
```

$\in$

```
element AUTHOR of type xs:string *
```

Laws — mapping XQuery into XQuery core

XPath slash and XQuery for

```
  /BOOKS/BOOK/AUTHOR
```

=

```
  let $root := / return
    for $dot1 in $root/BOOKS return
      for $dot2 in $dot1/BOOK return
        $dot2/AUTHOR
```

Laws — Associativity

Associativity in XPath

```
  BOOKS/(BOOK/AUTHOR)
=
  (BOOKS/BOOK)/AUTHOR
```

Associativity in XQuery

```
  for $dot1 in $root/BOOKS return
    for $dot2 in $dot1/BOOK return
      $dot2/AUTHOR
=
  for $dot2 in (
    for $dot1 in $root/BOOKS return
      $dot1/BOOK
  ) return
    $dot2/AUTHOR
```

Part I.5

Selection

Selection

Return titles of all books published before 2000

```
/BOOKS/BOOK[@YEAR < 2000]/TITLE
```

$\Rightarrow$

```
<TITLE>Data on the Web</TITLE>
```

$\in$

```
element TITLE of type xs:string *
```

Laws — mapping XQuery into XQuery core

Selection defined by where

```
  /BOOKS/BOOK[@YEAR < 2000]/TITLE
```

=

```
  for $book in /BOOKS/BOOK
  where $book/@YEAR < 2000
  return $book/TITLE
```

Selection defined by conditional

```
  for $book in /BOOKS/BOOK
  where $book/@YEAR < 2000
  returns $book/TITLE
```

=

```
  for $book in /BOOKS/BOOK returns
    if $book/@YEAR < 2000 then $book/TITLE else ()
```

Laws — mapping XQuery into XQuery core

Comparison defined by existential

```
  $book/@YEAR < 2000
```

$=$

```
  some $year in $book/@YEAR satisfies $year < 2000
```

Existential defined by iteration with selection

```
  some $year in $book/@YEAR satisfies $year < 2000
```

$=$

```
  not(empty(
    for $year in $book/@YEAR where $year < 2000 returns $year
  ))
```

Laws — mapping into XQuery core

```
  /BOOKS/BOOK[@YEAR < 2000]/TITLE
=
  let $root := / return
    for $books in $root/BOOKS return
      for $book in $books/BOOK return
        if (
          not(empty(
            for $year in $book/@YEAR returns
              if $year < 2000 then $year else ()
          ))
        ) then
          $book/TITLE
        else
          ()
```

Selection — Type may be too broad

Return book with title "Data on the Web"

```
  /BOOKS/BOOK[TITLE = "Data on the Web"]
```

⇒

```
  <BOOK YEAR="1999 2003">
    <AUTHOR>Abiteboul</AUTHOR>
    <AUTHOR>Buneman</AUTHOR>
    <AUTHOR>Suciu</AUTHOR>
    <TITLE>Data on the Web</TITLE>
    <REVIEW>A <EM>fine</EM> book.</REVIEW>
  </BOOK>
```

$\in$

```
   BOOK*
```

How do we exploit keys and relative keys?

Selection — Type may be narrowed

Return book with title "Data on the Web"

```
  treat as element BOOK? (
    /BOOKS/BOOK[TITLE = "Data on the Web"]
  )
```

$\in$

```
  BOOK?
```

Can exploit static type to reduce dynamic checking
Here, only need to check length of book sequence, not type

Iteration — Type may be too broad

Return all Amazon and BN books by Buneman

```
define element AMAZON-BOOK of type BOOK-TYPE
define element BN-BOOK of type BOOK-TYPE
define element CATALOGUE {
  element AMAZON-BOOK * , element BN-BOOK*
}

for $book in (/CATALOGUE/AMAZON-BOOK, /CATALOGUE/BN-BOOK)
where $book/AUTHOR = "Buneman"
return $book
```

$\in$

```
( element AMAZON-BOOK | element BN-BOOK )*
```

$\not\subseteq$

```
element AMAZON-BOOK * , element BN-BOOK *
```

How best to trade off simplicity vs. accuracy?

Part I.6

Construction

Construction in XQuery

Return year and title of all books published before 2000

```
for $book in /BOOKS/BOOK
where $book/@YEAR < 2000
return
  <BOOK>{ $book/@YEAR, $book/TITLE }</BOOK>
```

$\Rightarrow$

```
<BOOK YEAR="1999 2003">
  <TITLE>Data on the Web</TITLE>
</BOOK>
```

$\in$

```
element BOOK {
  attribute YEAR { integer+ },
  element TITLE { string }
} *
```

Construction — physical and logical

```
  <BOOK>{ $book/@YEAR , $book/TITLE }</BOOK>
=
  element BOOK { $book/@YEAR , $book/TITLE }

  <BOOK YEAR="{ data($book/@YEAR) }">
    <TITLE> data($book/TITLE) </TITLE>
  </BOOK>
=
  element BOOK {
    attribute YEAR { data($book/@YEAR) },
    element TITLE { data($book/TITLE) }
  }
```

Construction — attribute nodes

```
for $book in /BOOKS/BOOK
return
 <BOOK>
    if empty($book/@YEAR) then
      attribute YEAR  2000
    else
      $book/@YEAR ,
    $book/title
 </BOOK>
```

Part I.7

Grouping

Grouping

Return titles for each author

```
for $author in distinct(/BOOKS/BOOK/AUTHOR) return
  <AUTHOR NAME="{ $author }">{
    /BOOKS/BOOK[AUTHOR = $author]/TITLE
  }</AUTHOR>
```

$\Rightarrow$

```
<AUTHOR NAME="Abiteboul">
  <TITLE>Data on the Web</TITLE>
</AUTHOR>,
<AUTHOR NAME="Buneman">
  <TITLE>Data on the Web</TITLE>
  <TITLE>XML in Scotland</TITLE>
</AUTHOR>,
<AUTHOR NAME="Suciu">
  <TITLE>Data on the Web</TITLE>
</AUTHOR>
```

Grouping — Type may be too broad

Return titles for each author

```
for $author in distinct(/BOOKS/BOOK/AUTHOR) return
  <AUTHOR NAME="{ $author }">{
    /BOOKS/BOOK[AUTHOR = $author]/TITLE
  }</AUTHOR>
```

$\in$

```
element AUTHOR {
  attribute NAME { string },
  element TITLE { string } *
}
```

$\not\subseteq$

```
element AUTHOR {
  attribute NAME { string },
  element TITLE { string } +
}
```

Grouping — Type may be narrowed

Return titles for each author

```
define element TITLE { string }

for $author in distinct(/BOOKS/BOOK/AUTHOR) return
  <AUTHOR NAME="{ $author }">{
    treat as element TITLE+ (
      /BOOKS/BOOK[AUTHOR = $author]/TITLE
    )
  }</AUTHOR>
```

$\in$

```
element AUTHOR {
  attribute NAME { string },
  element TITLE { string } +
}
```

Part I.8

Join

Join

Books that cost more at Amazon than at BN

```
define element BOOKS {
  element BOOK *
}
define element BOOK {
  element TITLE of type xs:string ,
  element PRICE of type xs:decimal ,
  element ISBN of type xs:string
}

let $amazon := document("http://www.amazon.com/books.xml"),
    $bn := document("http://www.BN.com/books.xml")
for $a in $amazon/BOOKS/BOOK,
    $b in $bn/BOOKS/BOOK
where $a/ISBN = $b/ISBN
  and $a/PRICE > $b/PRICE
return <BOOK>{ $a/TITLE, $a/PRICE, $b/PRICE }</BOOK>
```

Join — Unordered

Books that cost more at Amazon than at BN, in any order

```
unordered(
  for $a in $amazon/BOOKS/BOOK,
      $b in $bn/BOOKS/BOOK
  where $a/ISBN = $b/ISBN
    and $a/PRICE > $b/PRICE
  return <BOOK>{ $a/TITLE, $a/PRICE, $b/PRICE }</BOOK>
)
```

Reordering required for cost-effective computation of joins

Join — Sorted

```
for $a in $amazon/BOOKS/BOOK,
    $b in $bn/BOOKS/BOOK
where $a/ISBN = $b/ISBN
 and $a/PRICE > $b/PRICE
order by $a/TITLE
return <BOOK>{ $a/TITLE, $a/PRICE, $b/PRICE }</BOOK>
```

Join — Laws

```
for $a in $amazon/BOOKS/BOOK,
    $b in $bn/BOOKS/BOOK
where $a/ISBN = $a/ISBN
  and $b/PRICE > $b/PRICE
order by $a/TITLE
return <BOOK>{ $a/TITLE, $a/PRICE, $b/PRICE }</BOOK>
```

=

```
for $x in
  unordered(
    for $a in $amazon/BOOKS/BOOK,
        $b in $bn/BOOKS/BOOK
    where $a/ISBN = $a/ISBN
      and $b/PRICE > $b/PRICE
    return <BOOK>{ $a/TITLE, $a/PRICE, $b/PRICE }</BOOK>
  )
order by $x/TITLE
return $x
```

Join — Laws

```
unordered(
  for $a in $amazon/BOOKS/BOOK,
      $b in $bn/BOOKS/BOOK
  where $a/ISBN = $a/ISBN
    and $a/PRICE > $b/PRICE
  return <BOOK>{ $a/TITLE, $a/PRICE, $b/PRICE }</BOOK>
)
=
unordered(
  for $a in unordered($amazon/BOOKS/BOOK),
      $b in unordered($bn/BOOKS/BOOK)
  where $a/ISBN = $a/ISBN
    and $b/PRICE > $b/PRICE
  return <BOOK>{ $a/TITLE, $a/PRICE, $b/PRICE }</BOOK>
)
```

Left outer join

Books at Amazon and BN with both prices,
and all other books at Amazon with price

```
for $a in $amazon/BOOKS/BOOK, $b in $bn/BOOKS/BOOK
where $a/ISBN = $b/ISBN
return <BOOK>{ $a/TITLE, $a/PRICE, $b/PRICE }</BOOK>
,
for $a in $amazon/BOOKS/BOOK
where not($a/ISBN = $bn/BOOKS/BOOK/ISBN)
return <BOOK>{ $a/TITLE, $a/PRICE }</BOOK>
```

$\in$

```
element BOOK { TITLE, PRICE, PRICE } *
,
element BOOK { TITLE, PRICE } *
```

Why type closure is important

Closure problems for Schema

- Deterministic content model
- Consistent element restriction

```
  element BOOK { TITLE, PRICE, PRICE } *
  ,
  element BOOK { TITLE, PRICE } *
```

$\subseteq$

```
  element BOOK { TITLE, PRICE+ } *
```

The first type is *not* a legal Schema type
The second type *is* a legal Schema type
Both are legal XQuery types

Part I.9

Nulls and three-valued logic

Books with price and optional shipping price

```
define element BOOKS { element BOOK * }
define element BOOK {
  element TITLE of type xs:string ,
  element PRICE of type xs:decimal ,
  element SHIPPING of type xs:decimal ?
}

<BOOKS>
  <BOOK>
    <TITLE>Data on the Web</TITLE>
    <PRICE>40.00</PRICE>
    <SHIPPING>10.00</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>XML in Scotland</TITLE>
    <PRICE>45.00</PRICE>
  </BOOK>
</BOOKS>
```

Approaches to missing data

Books costing \$50.00, where missing shipping is unknown

```
for $book in /BOOKS/BOOK
where $book/PRICE + $book/SHIPPING = 50.00
return $book/TITLE
```

$\Rightarrow$

```
<TITLE>Data on the Web</TITLE>
```

Books costing \$50.00, where default shipping is \$5.00

```
for $book in /BOOKS/BOOK
where $book/PRICE + ifAbsent($book/SHIPPING, 5.00) = 50.00
return $book/TITLE
```

$\Rightarrow$

```
<TITLE>Data on the Web</TITLE>,
<TITLE>XML in Scotland</TITLE>
```

Arithmetic, Truth tables

+	()	0	1
()	()	()	()
0	()	0	1
1	()	1	2

*	()	0	1
()	()	()	()
0	()	0	0
1	()	0	1

OR3	()	false	true
()	()	()	true
false	()	false	true
true	true	true	true

AND3	()	false	true
()	()	false	()
false	false	false	false
true	()	false	true

NOT3	
()	()
false	true
true	false

Part I.10

Type errors

Type error 1: Missing or misspelled element

Return TITLE and ISBN of each book

```
define element BOOK {
  element TITLE of type xs:string ,
  element PRICE of type xs:decimal ?
}

for $book in $books/BOOK return
  <ANSWER>{ $book/TITLE, $book/ISBN }</ANSWER>
```

$\in$

```
element ANSWER {
  element TITLE of type xs:string
} *
```

Finding an error by omission

Return title and ISBN of each book

```
define element BOOK {
  element TITLE of type xs:string ,
  element PRICE of type xs:decimal ?
}

for $book in /BOOKS/BOOK return
  <ANSWER>{ $book/TITLE, $book/ISBN }</ANSWER>
```

Report an error any sub-expression of type `()`, other than the expression `()` itself

Finding an error by assertion

Return title and ISBN of each book

```
define element BOOK {
  element TITLE of type xs:string ,
  element PRICE of type xs:decimal
}
define element ANSWER {
  element TITLE of type xs:string ,
  element ISBN of type xs:string
}

for $book in /BOOKS/BOOK return
  validate {
    <ANSWER>{ $book/TITLE, $book/ISBN }</ANSWER>
  }
```

Type Error 2: Improper type

```
define element BOOK {
  element TITLE of type xs:string ,
  element PRICE of type xs:decimal ,
  element SHIPPING of type xs:boolean ,
  element SHIPCOST of type xs:decimal ?
}

for $book in /BOOKS/BOOK return
  <ANSWER>{
     $book/TITLE,
     <TOTAL>{ $book/PRICE + $book/SHIPPING }</TOTAL>
  }</ANSWER>
```

Type error: `decimal + boolean`

Type Error 3: Unhandled null

```
define element BOOK {
  element TITLE of type xs:string ,
  element PRICE of type xs:decimal ,
  element SHIPPING of type xs:decimal ?
}
define element ANSWER {
  element TITLE of type xs:string ,
  element TOTAL of type xs:decimal
}

for $book in /BOOKS/BOOK return
  validate {
    <ANSWER>{
      $book/TITLE,
      <TOTAL>{ $book/PRICE + $book/SHIPPING }</TOTAL>
     }</ANSWER>
  }
```

Type error: $\mathrm{xsd : decimal?} \not\subseteq \mathrm{xsd : decimal}$

Part I.11

Functions

Functions

Simplify book by dropping optional year

```
define element BOOK { @YEAR?, AUTHOR, TITLE }
define attribute YEAR { xsd:integer }
define element AUTHOR { xsd:string }
define element TITLE { xsd:string }
define function simple (element BOOK $b) returns element BOOK {
  <BOOK> $b/AUTHOR, $b/TITLE </BOOK>
}
```

Compute total cost of book

```
define element BOOK { TITLE, PRICE, SHIPPING? }
define element TITLE { xsd:string }
define element PRICE { xsd:decimal }
define element SHIPPING { xsd:decimal }
define function cost (element BOOK $b) returns xsd:integer? {
  $b/PRICE + $b/SHIPPING
}
```

Part I.12

Recursion

A part hierarchy, with incremental costs

```
define element PART {
  attribute NAME of type xs:string &
  attribute COST of type xs:decimal ,
  element PART *
}

<PART NAME="system" COST="500.00">
  <PART NAME="monitor" COST="1000.00"/>
  <PART NAME="keyboard" COST="500.00"/>
  <PART NAME="pc" COST="500.00">
    <PART NAME="processor" COST="2000.00"/>
    <PART NAME="dvd" COST="1000.00"/>
  </PART>
</PART>
```

A recursive function, to compute total costs

```
define function total (element PART $part)
returns element PART {
  let $subparts := $part/PART/total(.)
  return
    <PART NAME="$part/@NAME"
          COST="$part/@COST + sum($subparts/@COST)">{
      $subparts
    }</PART>
}
```

Applying the function

```
  total(/PART)
```

$\Rightarrow$

```
  <PART NAME="system" COST="5000.00">
    <PART NAME="monitor" COST="1000.00"/>
    <PART NAME="keyboard" COST="500.00"/>
    <PART NAME="pc" COST="3500.00">
      <PART NAME="processor" COST="2000.00"/>
      <PART NAME="dvd" COST="1000.00"/>
    </PART>
  </PART>
```

Part I.13

Wildcard types

Wildcards types and computed names

Turn all attributes into elements, and vice versa

```
define function swizzle (element $x) returns element {
  element {name($x)} {
    for $a in $x/@* return element {name($a)} {data($a)},
    for $e in $x/* return attribute {name($e)} {data($e)}
  }
}

swizzle(<TEST A="a" B="b">
         <C>c</C>
         <D>d</D>
        </TEST>)
```

$\Rightarrow$

```
<TEST C="c" D="D">
  <A>a</A>
  <B>b</B>
</TEST>
```

$\in$

```
element
```

Part I.14

Syntax

Templates

Convert book listings to HTML format

```
<HTML><H1>My favorite books</H1>
  <UL>{
    for $book in /BOOKS/BOOK return
      <LI>
        <EM>{ data($book/TITLE) }</EM>,
        { data($book/@YEAR)[position()=last()] }.
      </LI>
  }</UL>
</HTML>
```

$\Rightarrow$

```
<HTML><H1>My favorite books</H1>
  <UL>
    <LI><EM>Data on the Web</EM>, 2003.</LI>
    <LI><EM>XML in Scotland</EM>, 2002.</LI>
  </UL>
</HTML>
```

XQueryX

A query in XQuery:

```
for $b in document("bib.xml")//book
where $b/publisher = "Morgan Kaufmann" and $b/year = "1998"
return $b/title
```

The same query in XQueryX:

```
<q:query xmlns:q="http://www.w3.org/2001/06/xqueryx">
 <q:flwr>
   <q:forAssignment variable="$b">
     <q:step axis="SLASHSLASH">
       <q:function name="document">
         <q:constant datatype="CHARSTRING">bib.xml</q:constant>
       </q:function>
       <q:identifier>book</q:identifier>
     </q:step>
   </q:forAssignment>
```

XQueryX, continued

```
<q:where>
  <q:function name="AND">
    <q:function name="EQUALS">
      <q:step axis="CHILD">
        <q:variable>$b</q:variable>
        <q:identifier>publisher</q:identifier>
      </q:step>
      <q:constant datatype="CHARSTRING">Morgan Kaufmann</q:consta
    </q:function>
    <q:function name="EQUALS">
      <q:step axis="CHILD">
        <q:variable>$b</q:variable>
        <q:identifier>year</q:identifier>
      </q:step>
      <q:constant datatype="CHARSTRING">1998</q:constant>
    </q:function>
  </q:function>
</q:where>
```

XQueryX, continued^2

```
    <q:return>
      <q:step axis="CHILD">
        <q:variable>$b</q:variable>
        <q:identifier>title</q:identifier>
      </q:step>
    </q:return>
  </q:flwr>
</q:query>
```

Part II

XPath and XQuery

XPath and XQuery

Converting XPath into XQuery core

e/a

$=$

`sidoaed(for $dot in` e `return $dot/`a`)`

sidoaed $=$ sort in document order and eliminate duplicates

Why sidoaed is needed

```
<WARNING>
  <P>
    Do <EM>not</EM> press button,
    computer will <EM>explode!</EM>
  </P>
</WARNING>
```

Select all nodes inside warning

```
/WARNING//*
```

⇒

```
<P>
  Do <EM>not</EM> press button,
  computer will <EM>explode!</EM>
</P>,
<EM>not</EM>,
<EM>explode!</EM>
```

Why sidoaed is needed, continued

Select text in all emphasis nodes (list order)

```
for $x in /WARNING//* return $x/text()
```

$\Rightarrow$

```
"Do ",
" press button, computer will ",
"not",
"explode!"
```

Select text in all emphasis nodes (document order)

```
/WARNING//*/text()
```

$=$

```
sidoaed(for $x in /WARNING//* return $x/text())
```

$\Rightarrow$

```
"Do ",
"not",
" press button, computer will ",
"explode!"
```

It’s life, Jim, but not as we know it

Parent

```
..
```

Find parents of all referee elements

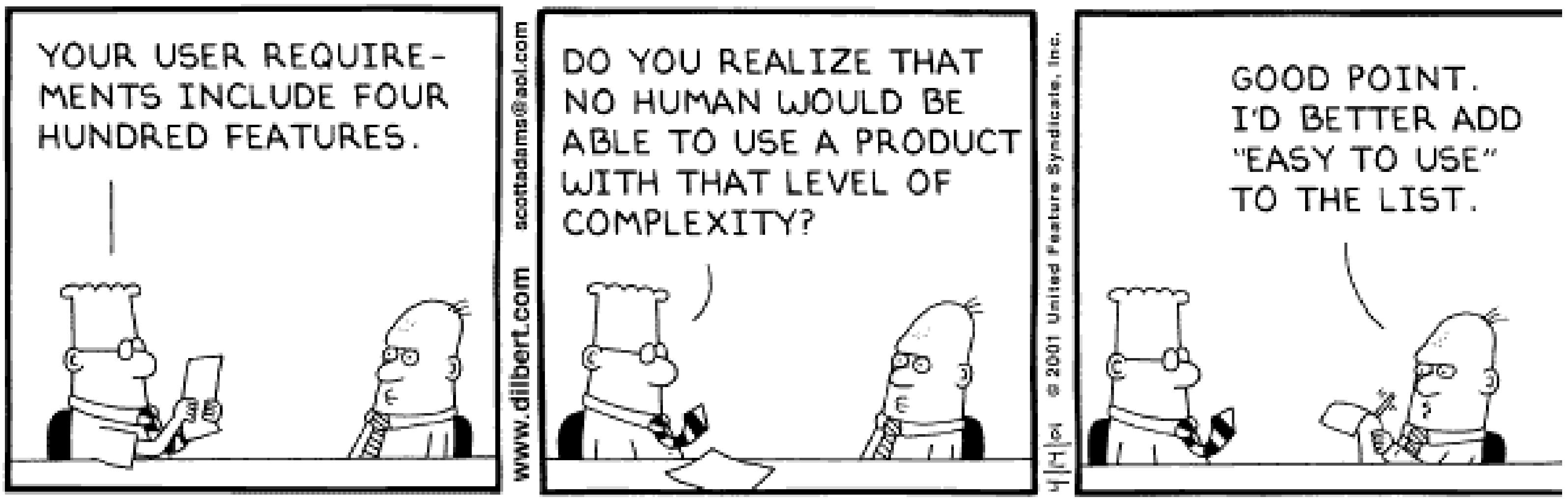
```
//referee/..
```

Naive implementation of element construction is quadratic!

Part III

DTD vs Schema vs XQuery

Dilbert


YOUR USER REQUIRE-MENTS INCLUDE FOUR HUNDRED FEATURES.
DO YOU REALIZE THAT NO HUMAN WOULD BE ABLE TO USE A PRODUCT WITH THAT LEVEL OF COMPLEXITY?
GOOD POINT. I'D BETTER ADD "EASY TO USE" TO THE LIST.
scottadams@aol.com
www.dilbert.com
© 2001 United Feature Syndicate, Inc.

Hilbert

"Besides it is an error to believe that rigor in the proof is the enemy of simplicity. On the contrary we find it confirmed by numerous examples that the rigorous method is at the same time the simpler and the more easily comprehended. The very effort for rigor forces us to find out simpler methods of proof."

— Hilbert

Expressive power - DTD

```
element BOOKS { element BOOK * }
element BOOK { element TITLE , element AUTHOR + }
element TITLE { xs:string }
element AUTHOR { xs:string }
```

Global definitions
Same element always has same content

Expressive power - Schema

```
element BOOKS {
  element AMAZON-BOOKS {
    element BOOK {
      element TITLE { xs:string } ,
      element AUTHOR { xs:string } +
    }
  }
  element BN-BOOKS {
    element BOOK {
      element AUTHOR { xs:string } + ,
      element TITLE { xs:string }
    }
  }
}
```

Nested definitions
Same element may have different content
Consistent sibling restriction

Expressive power - XQuery

```
element BOOKS {
  element BOOK {
    element TITLE { xs:string } ,
    element AUTHOR { xs:string } +
  }
  element BOOK {
    element AUTHOR { xs:string } + ,
    element TITLE { xs:string }
  }
}
```

Nested definitions
Same element may have different content
No consistent sibling restriction

Expressive power of XQuery types

Tree grammars and tree automata

	deterministic	non-deterministic
top-down		
bottom-up		

Tree grammar Class 0: DTD (global elements only)
Tree automata Class 1: Schema (determinism constraint)
Tree automata Class 2: XQuery, XDuce, Relax

$$\text{Class 0} < \text{Class 1} < \text{Class 2}$$

Class 0 and Class 2 have good closure properties.
Class 1 does not.

Expressive power of XQuery types

Tree grammars and tree automata

	deterministic	non-deterministic
top-down	Class 1	Class 2
bottom-up	Class 2	Class 2

Tree grammar Class 0: DTD (global elements only)
Tree automata Class 1: Schema (determinism constraint)
Tree automata Class 2: XQuery, XDuce, Relax

$$\text{Class } 0 < \text{Class } 1 < \text{Class } 2$$

Class 0 and Class 2 have good closure properties.
Class 1 does not.

Part IV

Type Inference

"I never come across one of Laplace's 'Thus it plainly appears' without feeling sure that I have hours of hard work in front of me."

— Bowditch

What is a type system?

- Validation: Value has type

$$v \in t$$

- Static semantics: Expression has type

$$e : t$$

- Dynamic semantics: Expression has value

$$e \Rightarrow v$$

- Soundness theorem: Values, expressions, and types match

$$\text{if} \quad e : t \quad \text{and} \quad e \Rightarrow v \quad \text{then} \quad v \in t$$

What is a type system? (with variables)

- Validation: Value has type

$$v \in t$$

- Static semantics: Expression has type

$$\bar{x} : \bar{t} \vdash e : t$$

- Dynamic semantics: Expression has value

$$\bar{x} \Rightarrow \bar{v} \vdash e \Rightarrow v$$

- Soundness theorem: Values, expressions, and types match

  if $\quad \bar{v} \in \bar{t} \quad$ and $\quad \bar{x} : \bar{t} \vdash e : t \quad$ and $\quad \bar{x} \Rightarrow \bar{v} \vdash e \Rightarrow v \quad$ then $\quad v \in t$

Documents

string	s	::=	`""` , `"a"`, `"b"`, `...`, `"aa"`, `...`	
integer	i	::=	`...`, `-1`, `0`, `1`, `...`	
document	d	::=	s	string
		|	i	integer
		|	`attribute` a `{` d `}`	attribute
		|	`element` a `{` d `}`	element
		|	`()`	empty sequence
		|	d `,` d	sequence

XQuery Types

unit type	u	$::=$	string	string
		|	integer	integer
		|	attribute a { t }	attribute
		|	attribute * { t }	wildcard attribute
		|	element a { t }	element
		|	element * { t }	wildcard element
type	t	$::=$	u	unit type
		|	()	empty sequence
		|	t , t	sequence
		|	t | t	choice
		|	t?	optional
		|	t+	one or more
		|	t*	zero or more
		|	x	type reference

Type of a document

- Overall Approach:
  Walk down the document tree

  Prove the type of d by proving the types of its constituent nodes.

- Example:

$$\frac{d \in t}{\texttt{element}\ a\ \texttt{\{}\ d\ \texttt{\}} \in \texttt{element}\ a\ \texttt{\{}\ t\ \texttt{\}}} \quad \text{(element)}$$

  Read: the type of `element` a `{` d `}` is `element` a `{` t `}` if the type of d is t.

Type of a document — $d \in t$

$$\frac{}{s \in \texttt{string}} \quad \text{(string)}$$

$$\frac{}{i \in \texttt{integer}} \quad \text{(integer)}$$

$$\frac{d \in t}{\texttt{element}\ a\ \texttt{\{}\ d\ \texttt{\}} \in \texttt{element}\ a\ \texttt{\{}\ t\ \texttt{\}}} \quad \text{(element)}$$

$$\frac{d \in t}{\texttt{element}\ a\ \texttt{\{}\ d\ \texttt{\}} \in \texttt{element}\ *\ \texttt{\{}\ t\ \texttt{\}}} \quad \text{(any element)}$$

$$\frac{d \in t}{\texttt{attribute}\ a\ \texttt{\{}\ d\ \texttt{\}} \in \texttt{element}\ a\ \texttt{\{}\ t\ \texttt{\}}} \quad \text{(attribute)}$$

$$\frac{d \in t}{\texttt{attribute}\ a\ \texttt{\{}\ d\ \texttt{\}} \in \texttt{element}\ *\ \texttt{\{}\ t\ \texttt{\}}} \quad \text{(any attribute)}$$

$$\frac{d \in t \qquad \texttt{define group}\ x\ \texttt{\{}\ t\ \texttt{\}}}{d \in x} \quad \text{(group)}$$

Type of a document, continued

$$\frac{}{\texttt{()} \in \texttt{()}} \qquad \text{(empty)}$$

$$\frac{d_1 \in t_1 \qquad d_2 \in t_2}{d_1 \texttt{,} d_2 \in t_1 \texttt{,} t_2} \qquad \text{(sequence)}$$

$$\frac{d_1 \in t_1}{d_1 \in t_1 \texttt{|} t_2} \qquad \text{(choice 1)}$$

$$\frac{d_2 \in t_2}{d_2 \in t_1 \texttt{|} t_2} \qquad \text{(choice 2)}$$

$$\frac{d \in t\texttt{+?}}{d \in t\texttt{*}} \qquad \text{(star)}$$

$$\frac{d \in t \texttt{,} t\texttt{*}}{d \in t\texttt{+}} \qquad \text{(plus)}$$

$$\frac{d \in \texttt{()} \texttt{|} t}{d \in t\texttt{?}} \qquad \text{(option)}$$

Type of an expression

- Overall Approach:
  Walk down the operator tree

  Compute the type of $expr$ from the types of its constituent expressions.

- Example:

$$\frac{e_1 \in t_1 \qquad e_2 \in t_2}{e_1\ ,\ e_2 \in t_1\ ,\ t_2} \qquad \text{(sequence)}$$

  Read: the type of e_1 , e_2 is a sequence of the type of e_1 and the type of e_2

Type of an expression — $E \vdash e \in t$

$$\text{environment } E \ ::= \ \$v_1 \in t_1, \ldots, \$v_n \in t_n$$

$$\frac{E \ \text{ contains } \ \$v \in t}{E \vdash \$v \in t} \qquad \text{(variable)}$$

$$\frac{E \vdash e_1 \in t_1 \qquad E, \$v \in t_1 \vdash e_2 \in t_2}{E \vdash \texttt{let } \$v \texttt{ := } e_1 \texttt{ return } e_2 \in t_2} \qquad \text{(let)}$$

$$\frac{}{E \vdash \texttt{()} \in \texttt{()}} \qquad \text{(empty)}$$

$$\frac{E \vdash e_1 \in t_1 \qquad E \vdash e_2 \in t_2}{E \vdash e_1 \texttt{ , } e_2 \in t_1 \texttt{ , } t_2} \qquad \text{(sequence)}$$

$$\frac{E \vdash e \in t_1 \qquad t_1 \cap t_2 \neq \emptyset}{E \vdash \texttt{treat as } t_2 \ \texttt{(}e\texttt{)} \in t_2} \qquad \text{(treat as)}$$

$$\frac{E \vdash e \in t_1 \qquad t_1 \subseteq t_2}{E \vdash \texttt{assert as } t_2 \ \texttt{(}e\texttt{)} \in t_2} \qquad \text{(assert as)}$$

Typing FOR loops

Return all Amazon and BN books by Buneman

```
define element AMAZON-BOOK { TITLE, AUTHOR+ }
define element BN-BOOK { AUTHOR+, TITLE }
define element BOOKS { AMAZON-BOOK*, BN-BOOK* }
for $book in (/BOOKS/AMAZON-BOOK, /BOOKS/BN-BOOK)
where $book/AUTHOR = "Buneman" return
  $book
```

$\in$

```
( AMAZON-BOOK | BN-BOOK )*
```

$$\frac{\begin{array}{c} E \vdash e_1 \in t_1 \\ E,\ \$x \in \mathsf{P}(t_1) \vdash e_2 \in t_2 \end{array}}{E \vdash \texttt{for}\ \$x\ \texttt{in}\ e_1\ \texttt{return}\ e_2 \in t_2 \cdot \mathsf{Q}(t_1)} \quad \text{(for)}$$

$$\begin{array}{rcl} \mathsf{P}(\texttt{AMAZON-BOOK*,BN-BOOK*}) & = & \texttt{AMAZON-BOOK | BN-BOOK} \\ \mathsf{Q}(\texttt{AMAZON-BOOK*,BN-BOOK*}) & = & * \end{array}$$

Prime types

unit type	u	$::=$	`string`	string
		$\mid$	`integer`	integer
		$\mid$	`attribute` a `{` t `}`	attribute
		$\mid$	`attribute * {` t `}`	any attribute
		$\mid$	`element` a `{` t `}`	element
		$\mid$	`element * {` t `}`	any element
prime type	p	$::=$	u	unit type
		$\mid$	p `|` p	choice

Quantifiers

quantifier q	::=	()	exactly zero		$t \cdot$ ()	=	()
	|	-	exactly one		$t \cdot$ -	=	t
	|	?	zero or one		$t \cdot$?	=	t?
	|	+	one or more		$t \cdot$ +	=	t+
	|	*	zero or more		$t \cdot$ *	=	t*

,	()	-	?	+	*
()	()	-	?	+	*
-	-	+	+	+	+
?	?	+	*	+	*
+	+	+	+	+	+
*	*	+	*	+	*

|	()	-	?	+	*
()	()	?	?	*	*
-	?	-	?	+	*
?	?	?	?	*	*
+	*	+	*	+	*
*	*	*	*	*	*

$\cdot$	()	-	?	+	*
()	()	()	()	()	()
-	()	-	?	+	*
?	()	?	?	*	*
+	()	+	*	+	*
*	()	*	*	*	*

$\leq$	()	-	?	+	*
()	$\leq$		$\leq$		$\leq$
-		$\leq$	$\leq$	$\leq$	$\leq$
?			$\leq$		$\leq$
+				$\leq$	$\leq$
*					$\leq$

Factoring

$$\begin{array}{lcl@{\qquad}lcl}
\mathsf{P}'(u) & = & \{u\} & \mathsf{Q}(u) & = & \texttt{-} \\
\mathsf{P}'(()) & = & \{\} & \mathsf{Q}(()) & = & () \\
\mathsf{P}'(t_1\ ,\ t_2) & = & \mathsf{P}'(t_1) \cup \mathsf{P}'(t_2) & \mathsf{Q}(t_1\ ,\ t_2) & = & \mathsf{Q}(t_1)\ ,\ \mathsf{Q}(t_2) \\
\mathsf{P}'(t_1 \mid t_2) & = & \mathsf{P}'(t_1) \cup \mathsf{P}'(t_2) & \mathsf{Q}(t_1 \mid t_2) & = & \mathsf{Q}(t_1) \mid \mathsf{Q}(t_2) \\
\mathsf{P}'(t?) & = & \mathsf{P}'(t) & \mathsf{Q}(t?) & = & \mathsf{Q}(t) \cdot ? \\
\mathsf{P}'(t+) & = & \mathsf{P}'(t) & \mathsf{Q}(t+) & = & \mathsf{Q}(t) \cdot + \\
\mathsf{P}'(t*) & = & \mathsf{P}'(t) & \mathsf{Q}(t*) & = & \mathsf{Q}(t) \cdot *
\end{array}$$

$$\begin{array}{lcll}
\mathsf{P}(t) & = & () & \text{if } \mathsf{P}'(t) = \{\} \\
 & = & u_1 \mid \cdots \mid u_n & \text{if } \mathsf{P}'(t) = \{u_1, \ldots, u_n\}
\end{array}$$

Factoring theorem. For every type t, prime type p, and quantifier q, we have $t \subseteq p \cdot q$ iff $\mathsf{P}(t) \subseteq p?$ and $\mathsf{Q}(t) \leq q$.

Corollary. For every type t, we have $t \subseteq \mathsf{P}(t) \cdot \mathsf{Q}(t)$.

Uses of factoring

$$\frac{E \vdash e_1 \in t_1 \qquad E, \texttt{\$}x \in \mathsf{P}(t_1) \vdash e_2 \in t_2}{E \vdash \texttt{for \$}x\texttt{ in } e_1 \texttt{ return } e_2 \in t_2 \cdot \mathsf{Q}(t_1)} \quad (\text{for})$$

$$\frac{E \vdash e \in t}{E \vdash \texttt{unordered(}e\texttt{)} \in \mathsf{P}(t) \cdot \mathsf{Q}(t)} \quad (\text{unordered})$$

$$\frac{E \vdash e \in t}{E \vdash \texttt{distinct(}e\texttt{)} \in \mathsf{P}(t) \cdot \mathsf{Q}(t)} \quad (\text{distinct})$$

$$\frac{\begin{array}{ll} E \vdash e_1 \in \texttt{integer} \cdot q_1 & q_1 \leq \texttt{?} \\ E \vdash e_2 \in \texttt{integer} \cdot q_2 & q_2 \leq \texttt{?} \end{array}}{E \vdash e_1 \texttt{ + } e_2 \in \texttt{integer} \cdot q_1 \cdot q_2} \quad (\text{arithmetic})$$

Subtyping and type equivalence

Definition. Write $t_1 \subseteq t_2$ iff for all d, if $d \in t_1$ then $d \in t_2$.

Definition. Write $t_1 = t_2$ iff $t_1 \subseteq t_2$ and $t_2 \subseteq t_1$.

Examples

$$t \subseteq t\texttt{?} \subseteq t\texttt{*}$$
$$t \subseteq t\texttt{+} \subseteq t\texttt{*}$$
$$t_1 \subseteq t_1 \mid t_2$$
$$t \texttt{ , } () = t = () \texttt{ , } t$$
$$t_1 \texttt{ , } (t_2 \mid t_3) = (t_1 \texttt{ , } t_2) \mid (t_1 \texttt{ , } t_3)$$
$$\texttt{element } a \texttt{ \{ } t_1 \mid t_2 \texttt{ \} } = \texttt{element } a \texttt{ \{ } t_1 \texttt{ \} } \mid \texttt{element } a \texttt{ \{ } t_2 \texttt{ \}}$$

Can decide whether $t_1 \subseteq t_2$ using tree automata:
$Language(t_1) \subseteq Language(t_2)$ iff
$Language(t_1) \cap Language(Complement(t_2)) = \emptyset$.

Part V

The Essence of XML

Named typing

Schema

```
<xs:simpleType name="feet">
  <xs:restriction base="xs:integer"/>
</xs:simpleType>
<xs:element name="height" type="feet"/>
```

XQuery

```
define type feet restricts xs:integer
define element height of type feet
```

Validation

Document in XML

```
<height>10023</height>
```

Data model, before validation

```
  <height>10023</height>
⇒
  element height { "10023" }
```

Data model, after validation

```
  validate as element height { <height>10023</height> }
⇒
  element height of type feet { 10023 }
```

Matching

Unvalidated data does not match

  element height { "10023" }

matches

  element height of type feet

(NOT!)

Validated data may match

  element height of type feet { 10023 }

matches

  element height of type feet

Erasure

The inverse of validation is type erasure

element height of type feet { 10023 }

erases to

element height { "10023" }

Erasure is a relation

validate as xs:integer ("7") $\Rightarrow$ 7

validate as xs:integer ("007") $\Rightarrow$ 7

7 **erases to** "7"

7 **erases to** "007"

The validation theorem

Theorem *We have that*

$$\textbf{validate as } \mathit{Type} \; \{ \; \mathit{UntypedValue} \; \} \Rightarrow \; \mathit{Value}$$

if and only if

$$\mathit{Value} \; \textbf{matches} \; \mathit{Type}$$

and

$$\mathit{Value} \; \textbf{erases to} \; \mathit{UntypedValue}.$$

Not as obvious as it looks! Key is that erasure is a relation

Part VI

Further reading and experimenting

Related work

Xduce — Haruo Hasoya and Benjamin Pierce

RelaxNG — James Clark and Makoto Murata

Links

Phil's XML page

`http://www.research.avayalabs.com/~wadler/xml/`

W3C XML Query page

`http://www.w3.org/XML/Query.html`

XML Query demonstrations

```
Galax - AT&T, Lucent, and Avaya
  http://www-db.research.bell-labs.com/galax/
Quip - Software AG
  http://www.softwareag.com/developer/quip/
XQuery demo - Microsoft
  http://131.107.228.20/xquerydemo/
Fraunhofer IPSI XQuery Prototype
  http://xml.ipsi.fhg.de/xquerydemo/
XQengine - Fatdog
  http://www.fatdog.com/
X-Hive
  http://217.77.130.189/xquery/index.html
OpenLink
  http://demo.openlinksw.com:8391/xquery/demo.vsp
```