XPath

→ Query language to select (a sequence of) nodes of an XML document
→ W3C Standard
→ Most important XML query language: used in many other standards such as XQuery, XSLT, XPath, XLink, ...
→ Cave: version 2.0 is considerably more expressive than 1.0
  We study XPath 1.0

Terminology: Instead of XPath "query" we often say XPath expression.
(An expression is the primary construction of the XPath grammar; it matches the production Expr of the XPath grammar.)

XPath Data Model

Evaluate Q on D (in XPath data model)

XPath Query Q
XML document D

sequence of result nodes

Document D is modeled as a tree.
THERE ARE SEVEN TYPES OF NODES in the XPath Data Model:

root nodes
element nodes
text nodes
attribute nodes
namespace nodes
processing instruction nodes
comment nodes

Outline - Lectures

1. Introduction to XML, Encodings, Parsers
2. Memory Representations for XML: Space vs Access Speed
3. RDBMS Representation of XML
4. DTDs, Schemas, Regular Expressions, Ambiguity
5. XML Validation using Automata
6. Node Selecting Queries: XPath
7. Tree Automata for Efficient XPath Evaluation, Parallel Evaluation
8. XPath Properties: backward axes, containment test
9. Streaming Evaluation: how much memory do you need?
10. XPath Evaluation using RDBMS
11. XSLT – stylesheets and transform
12. XQuery – XML query language

Outline - Assignments

2. SAX Parse into memory structure: Tree and DAG
3. Map XML into RDBMS → 20. April
4. XPath evaluation → 11. May
5. XPath into SQL Translation → 25. June

Outline

1. XPath Data Model: 7 types of nodes
2. Simple Examples
3. Location Steps and Paths
4. Value Comparison, and Other Functions
XPath Data Model

Document D is modeled as a tree.

THERE ARE SEVEN TYPES OF NODES in the XPath Data Model:

- root nodes
- element nodes
- text nodes
- attribute nodes
- namespace nodes
- processing instruction nodes
- comment nodes

For rest of lecture: this is ALL you need to know about XML nodes!

5.2.1 Unique IDs

An element node may have a unique identifier (ID).
- Value of the attribute that is declared in the DTD as type ID.
- No two elements in a document may have the same unique ID.
- If an XML processor reports two elements in a document as having the same unique ID (which is possible only if the document is invalid) then the second element in doc. order must be treated as not having a unique ID.

NOTE: If a document has no DTD, then no element will have a unique ID.

XPath Result Sequences

Evaluate Q on D (in XPath data model)

XPath Query Q

XML document D

sequence of result nodes

Ordered in document order
Contains no duplicates

Simple Examples

In abbreviated XPath syntax.

Q0: /

Selects the document root (always the parent of the document element)

Note: XPath Evaluators usually return the full subtree of the selected node.
Simple Examples

In abbreviated syntax:
Q1: /bib/book/year

Document:
<bib>
  <book>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <title>Foundations of Databases</title>
    <year>1995</year>
  </book>
  <book>
    <author>Ullmann</author>
    <title>Principles of Database and Knowledge Base Systems</title>
    <year>1998</year>
  </book>
</bib>

Result of query Q1:
- descendant or self nodes relative to the context-node = root node
- child nodes that are labeled year

// is short for /descendant-or-self::node().
For example, //author is short for /descendant-or-self::node()/child::author

Simple Examples

In abbreviated syntax:
Q2: //author

Document:
<bib>
  <book>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <title>Foundations of Databases</title>
    <year>1995</year>
  </book>
  <book>
    <author>Ullmann</author>
    <title>Principles of Database and Knowledge Base Systems</title>
    <year>1998</year>
  </book>
</bib>

Result of query Q2:
sequence of (element) nodes (N1, N2, N3, N4)

Simple Examples

In abbreviated syntax:
Q3: /a/b//d

d

Document:
<book>
  <author>Abiteboul</author>
  <author>Hull</author>
  <author>Vianu</author>
  <title>Foundations of Databases</title>
  <year>1995</year>
</book>

Result of query Q3:
ALL d-nodes in this subtree

// is short for /descendant-or-self::node().
For example, //author is short for /descendant-or-self::node()/child::author

Simple Examples

In abbreviated syntax:
Q4: /*b*/d

Document:
<book>
  <author>Abiteboul</author>
  <author>Hull</author>
  <author>Vianu</author>
  <title>Foundations of Databases</title>
  <year>1995</year>
</book>

Result of query Q4:
ALL d-nodes in this subtree
Simple Examples
In abbreviated syntax.
Q4: /*/c

Abbreviations, so far
In abbreviated syntax.
/a is abbreviation for /child::a
//a is abbreviation for /descendant-or-self::node()/child::a

Examples: Predicates
In abbreviated syntax.
Q7: ///c[./b]

20
21
22
23
24
Examples: Predicates

In abbreviated syntax.

Q9: //c[./b]/d  "has b-child"

Q9 selects c-nodes that "have a b-child AND a d-child"

More direct way: //c[./b and ./d]

We do not read "/.b"  \textit{self::node} /child::b  equivalent to \textit{b}
Examples: Predicates (or “Filters”)

In abbreviated syntax:

```xml
//c[b and d]
```

A “Filter”

How to only select the other `c`-node?

Can use “not( … )” in a filter!

```xml
//c[not(b)]
```

“does not have a b-child”

Question

How to only select the other `c`-node?

Many more possibilities, of course:

```xml
//c[parent::b]
//c[../../b]
//c[../d]
```

CAVE: what does ```//c[../b]``` give??
Examples: Predicates

In abbreviated syntax.

$$\forall \text{c}!: \neg \text{b}$$

same as for parent selection (\text{\textendash}..)

```
\forall \text{c}!: \neg \text{b} \lor \text{c}!: \neg (\text{child}::* \neg \neg \text{self}::\text{d})
```

"not the case that all children are not labeled d"
holds if and only if
"all children are labeled d"

Duplicate elimination

\text{c}!: \neg \text{b} / \text{d}/\text{ancestor}::*

\text{context-nodes}

Equivalent one, without use of \text{\textendash}ancestor\text{??}

```
\forall \text{c}!: \neg \text{b} \lor \neg (\text{child}::* \neg \neg \text{self}::\text{d})
```

\text{\textendash}ancestor::*

\text{context-nodes}

No use of \text{\textendash}ancestor\text{??}

```
\forall \text{c}!: \neg \text{b} \lor \neg (\text{child}::* \neg \neg \text{self}::\text{d}) \land \text{\textendash}ancestor::*
```

"only d-children" "has child (not leaf)"

How to select the c-node?

```
\forall \text{c}!: \neg \text{b} \land \text{\\textendash}ancestor::*
```

No use of \text{\textendash}ancestor\text{??}

```
\forall \text{c}!: \neg \text{b} \land \neg (\text{child}::* \neg \neg \text{self}::\text{d}) \land \text{\textendash}ancestor::*
```

"only d-children" "has child (not leaf)"

How to select the c-node?
More Details

Evaluate Q on D (in XPath data model)

sequence of result nodes

NOT correct (at least not for intermediate expr’s)

An expression evaluates to an object, which has one of the following four basic types:

- node-set: (an unordered collection of nodes w/o duplicates)
- boolean: (true or false)
- number: (a floating-point number)
- string: (a sequence of UCS characters)

Location Steps & Paths

A Location Path is a sequence of Location Steps

Initial Context will be root node

Location Paths

1. LocationPath := RelativeLocationPath

2. AbsoluteLocationPath := RelativeLocationPath * Step

3. RelativeLocationPath := Step

4. Step := AxisSpecifier NodeTest Predicate

5. AxisSpecifier := AxisName

Location Steps

A Location Step is of the form

axis :: nodetest [ Filter_1 ] [ Filter_2 ] … [ Filter_n ]

Filters (aka predicates, (filter) expressions)

evaluate to true/false

XPath queries, evaluated with context-node = current node

Boolean operators: and, or

Empty string/sequence are converted to false

Location Steps & Paths

A Location Path is a sequence of Location Steps

A Location Step is of the form

axis :: nodetest [ Filter_1 ] [ Filter_2 ] … [ Filter_n ]

Filters (aka predicates, (filter) expressions)

evaluate to true/false

nodetest: * or node-name (could be expanded xmlns)

Example: child::text() “select all text node children of the context node”

attribute:: “select all attributes of the context node”

Axis = a sequence of nodes (is evaluated relative to context-node)

12 Axes

Forward Axes: Backward Axes:

- self  →  parent
- child  →  ancestor
- descendant  →  ancestor-or-self
- following  →  preceding
- following-sibling  →  attribute
Location Steps & Paths

Axis = a sequence of nodes (is evaluated relative to context-node)

Forward Axes:
- self
- child
- descendant-or-self
- descendant
- following
- following-sibling

Backward Axes:
- parent
- ancestor
- ancestor-or-self
- preceding
- preceding-sibling
- attribute

In doc order: reverse doc order
Location Steps & Paths

Axis = a sequence of nodes (is evaluated relative to context-node)

Forward Axes:
- self
- child
- descendant-or-self
- descendant
- following
- following-sibling

Backward Axes:
- parent
- ancestor
- ancestor-or-self
- preceding
- preceding-sibling
- attribute

In doc order
Reverse doc order

Axis = a sequence of nodes (is evaluated relative to context-node)

Forward Axes:
- self
- child
- descendant-or-self
- descendant
- following
- following-sibling

Backward Axes:
- parent
- ancestor
- ancestor-or-self
- preceding
- preceding-sibling
- attribute

In doc order
Reverse doc order

Axis = a sequence of nodes (is evaluated relative to context-node)

Forward Axes:
- self
- child
- descendant-or-self
- descendant
- following
- following-sibling

Backward Axes:
- parent
- ancestor
- ancestor-or-self
- preceding
- preceding-sibling
- attribute

In doc order
Reverse doc order

Axis = a sequence of nodes (is evaluated relative to context-node)

Forward Axes:
- self
- child
- descendant-or-self
- descendant
- following
- following-sibling

Backward Axes:
- parent
- ancestor
- ancestor-or-self
- preceding
- preceding-sibling
- attribute

In doc order
Reverse doc order

Axis = a sequence of nodes (is evaluated relative to context-node)

Forward Axes:
- self
- child
- descendant-or-self
- descendant
- following
- following-sibling

Backward Axes:
- parent
- ancestor
- ancestor-or-self
- preceding
- preceding-sibling
- attribute

In doc order
Reverse doc order
Location Path Evaluation

Context of an XPath evaluation:

1. context-node
2. context position and size (both non-negative integers)
3. set of variable bindings (= mappings from variable names to values)
4. function library (= mapping from function names to functions)
5. set of namespace declarations

(bt: context position is \(\leq\) context size)

Application determines the Initial Context.

If path starts with "/", then Initial Context has

- context-node = root node
- context-position = context-size = 1

Location Path Semantics

\(\rightarrow\) A Location Path \(P\) is a sequence of Location Steps

\[ a_1 : n_1[F_1_1]F_1_2[...][F_1_n_1] \]/\[ a_2 : n_2[F_2_1]F_2_2[...][F_2_n_2] \]/\[ a_m : n_m[F_m_1]F_m_2[...][F_m_n_m] \]

\(S_0 =\) initial sequence of context-nodes

1. (to each) context-node \(N\) in \(S_0\), apply axis \(a_1\) gives sequence \(S_1\) of nodes
2. remove from \(S_1\) any node \(M\) for which
   - \(n_1\) evaluates to false
   - any of \(F_1_1,...,F_1_n_1\) evaluate to false.

Apply steps (1) & (2) for step 2, to obtain from \(S_1\) the sequence \(S_2\)

\[ S_3 \]
\[ ... \]
\[ S_{m-1} \]
\[ S_m \]

= result of \(P\)

No Looking Back

Backward Axes are not needed!!

\(\rightarrow\) possible to rewrite every XPath query into an equivalent one that does not use backward axes.

Very nice result! ☺

Can you see how this could be done?

\(\rightarrow\) We saw an example of removing ancestor axis. But, of course the rewritten query must be the same ON EVERY possible tree!!

Questions how much larger does the query get, when you remove all backward axis?

Is this useful for efficient query evaluation?!

Attribute Axis

Examples

//attribute::*

Result:

\(<b a="1">d</b>\>
\(<c a="1.0">d</c>\>
\(<d/>\>

\(\rightarrow\) test attribute nodes

\(\rightarrow\) Examples

\(/attribute::a=1\) gives same result and

\(\rightarrow\) for \(\rightarrow\) attribute::a gives

\(<d/>\>
\(<c a="1">a</c>\>
\(<c a="1">a</c>\>
\(<c a="1.0">a</c>\>

\(\rightarrow\) Watch out

\(/attribute::a=1\) only gives

\(\rightarrow\) number (float) comparison

\(/attribute::a=1.0\) only gives

\(\rightarrow\) string comparison
### Attribute Axis & Value Tests

**How to test attribute values**
- Examples
  - `/*[attribute::a=1]` (selects the two red nodes)

**Watch out**
- `/*[attribute::a="1"]` only gives
- `/*[attribute::a="1.0"]` only gives

*attribute:: is abbreviated by @*

### Tests in Filters

- or
- and
- `<`, `<=`, `>`, `>=`

The operators are all left associative. For example, `3 > 2 > 1` is equivalent to `(3 > 2) > 1`, which evaluates to `false`.

But, `3 > 2 > 0.9` evaluates to `true`. Can you see why?

For two strings `u,v`

- `u<=v` 
- `u<v`
- `u>=v`
- `u>v`

Always return `false`!

- Unless both `u` and `v` are numbers.
- `["1.0">="1"]` evaluates to `true`.

### Text Nodes

**How to test text nodes & values**
- Examples
  - `//text()`

**Result**:
- `foo`
- `foo`
- `Bar`

- `/*[text()="foo"]` Result: the two red nodes

**Question**:

- What is the result for
  - `//text()[count(//*[text()=//b/text()])=2]`

### Useful Functions (on Booleans)

- `boolean(object):boolean` *("boolean" means `true/false`)*
  - Converts argument into `true/false`:
    - a number is `true` if it is not equal to zero (or NaN)
    - a node-set is `true` if it is non-empty
    - a string is `true` if its length is non-zero
  - for other objects, conversion depends on type

- `not(true)=false, not(false)=true`
- `true(); boolean`
- `false(); boolean`
- `lang(string): boolean`

Returns `true` if language specified by `xml:lang` attributes is same as string

Useful even for use with self-axis:

- `child::*[self::chapter or self::appendix]`
- `chapter or appendix` children of context node
Useful Functions (on Node Sets)

- `count`
  Counts number or results

- `last()`
  Returns context-size from the evaluation context

- `position()`
  Returns context-position from the evaluation context

What is the result?

Same result as:

```
/a[count(//*[text()="foo"])]
> count(//*[text()="bar"])]
```

What is the result for:

```
//c[position()=0]
```

(same as `/c[count(b)=0]`)

```
//*[position()=2]
```

```
//*[position()=2 and ../../a]
```

Same as

```
//*[position()=2 and ./b]
```

```
//*[position()=last()-1]
```
Useful Functions (on Node Sets)

- `last()` returns context-size from the evaluation context.
- `position()` returns context-position from the evaluation context.

Example: Selecting the last two book-children of books:

```
//*[position()=1]/*[position()=2]/*[position()=2]
```

Abbreviation: `/*[1]/*[2]/*[2]`

What is result for `//*[./*[2]/*[2]]`?
Useful Functions (on Node Sets)

Æ last():number
returns context-size from the evaluation context

Æ position():number
returns context-position from the eval. context

books

How do you select the last 20 book-children of books?

last 20 /books/book[position()>last()-20]

Useful Functions (on Node Sets)

Nodes have an identity

æ Different nodes!

/(a)[1]=*[2]

But:

//a[contains(*[1],*[2])]
gives the a-node.

gives the a-node. string-value ("It") is contained in "It"

Useful Functions (on Node Sets)

Careful with equality ("=")

æ What about //a[b/d=c/d] selects a-node!!

there exists a node in the node set for b/d with same string value as a node in node set c/d

 æ What about //a[b/d = c/d]

 æ What about //a[b/d = c/d]

æ C = 1.0
æ D = "foo"
æ E = "fobar"
æ F = "ba"
Useful Functions (Strings)

The string-value of an element node is the concatenation of the string-values of all text node descendants in document order.

\[ \text{concat}(s_1, s_2, \ldots, s_n) = s_1 s_2 \ldots s_n \]

\[ \text{startswith}("abcd","ab") = \text{true} \]

\[ \text{contains}("bar","a") = \text{true} \]

\[ \text{substring}-before("1999/04/01","/" ) = 1999 \]

\[ \text{substring}-after("1999/04/01","19") = 99/04/01 \]

\[ \text{substring}("12345",2,3) = "234" \]

\[ \text{string-length}("foo") = 3 \]

What is the result to this: 

\[ \text{//}*[="foo"] \]

\[ \text{//}*[="foobar"] \]

\[ \text{normalize-space}(" foo   bar a ") = "foo bar a" \]

\[ \text{translate}("bar","abc","ABC") = \text{BAr} \]

returns the first argument string with occurrences of characters in the second argument string replaced by the character at the corresponding position in the third argument string.

**NOTE:** The `translate` function is not a sufficient solution for case conversion in all languages.

Useful Functions (Numbers)

\[ \text{number}(\text{object}):\text{number} \]

Converts argument to a number:

- the boolean true is converted to 1, false is converted to 0
- a string that consists of optional whitespace followed by an optional minus sign followed by a Number followed by whitespace is converted to the IEEE 754 number that is nearest to the mathematical value represented by the string.

\[ \text{sum}(\text{node-set}):\text{number} \]

returns sum, for each node in the argument node-set, of the result of converting the string-value of the node to a number.

\[ \text{floor}(\text{number}):\text{number} \]

returns largest integer that is not greater than the argument.

\[ \text{ceiling}(\text{number}):\text{number} \]

returns the smallest integer that is not less than the argument.

\[ \text{round}(\text{number}):\text{number} \]

returns integer closest to the argument. (If there are 2, take above: \( \text{round}(0.5) = 1 \) and \( \text{round}(-0.5) = 0 \)

Operators on Numbers

+,-,*,/,

Display Number Result:

\[ \text{//}*[text()=7 \mod (\text{count}(//b)+2)]/text() \]

Similar for arbitrary large numbers / booleans, node-sets... Try it...

XPath Query Evaluation

How to implement?

How expensive? complexity?

What are the most difficult queries?

Next time

Efficient Algorithms: which queries run how fast?

First, focus on navigational queries: only /, //, label-test; [ filters ]

(tenics for value comparison/queries already well-known from rel. DB's...)

Display Number Result:

\[ \text{//}*[text()=7 \mod (\text{count}(//b)+2)]/text() \]

Use: [http://b-cage.net/code/web/xpath-evaluator.html](http://b-cage.net/code/web/xpath-evaluator.html)
Experiments with current systems

Next 4 slides from Georg Gottlob and Christoph Koch *XPath Query Processing*.
Invited tutorial at DBPL 2003
http://www.dbai.tuwien.ac.at/research/xmltaskforce/xpath-tutorial1.ppt.gz

XPath Query (relative to a):
child::*/parent::*/child::*
parent::*/child::*

Tree of nodes visited is of size $O(D^6)$ !!!

Core XPath on Xalan and XT
Queries: a/b/parent::a/.../parent::a/b

Core XPath on Microsoft IE6:
polynomial combined complexity, quadratic data complexity

Full XPath on IE6:
Exponential combined complexity!
Exponential query complexity

XPath Query Evaluation

Static Methods (used, e.g., for Query Optimization...)

Given XPath queries Q1, Q2:
→ Is result set of Q1 included in result set of Q2?
→ Are result sets equal?
→ Is their intersection empty?

for all possible documents

(probably we will look at this in Lecture 8 or 9)
Simple Examples

Is

//c[count(d)=count(*)]

equivalent to

//c[not(child::*[not(self::d)])]
on all possible trees?

END

Lecture 6