Outline

→ Lecture 19 (Monday March 27th): Recap / Exam preparation

→ Lecture 20 (Monday March 27th): Recap / Exam preparation

→ no lectures after that! [i.e., no lectures on April 3 & 6]
Outline

1. XPath

2. XSLT – eXtensible Stylesheet Language Transformations
Location Steps & Paths

A Location Path is a sequence of Location Steps

A Location Step is of the form

\[ \text{axis} :: \text{nodetest} \ [ \text{Filter}_1 \] \ [ \text{Filter}_2 \] \ldots \ [ \text{Filter}_n \] \]

Filters (aka predicates, (filter) expressions) evaluate to true/false

\[ \text{nodetest:} \ * \text{ or node-name} \ (\text{could be expanded to namespaces}) \text{ or} \]

12 Axes

Forward Axes:

- self
- child
- descendant-or-self
- descendant
- following
- following-sibling

In doc order

Backward Axes:

- parent
- ancestor
- ancestor-or-self
- preceding
- preceding-sibling

\[ \text{reverse doc order} \]

\[ \text{node()} \]
1. XPath

Query that selects c-nodes that have a b-child?
1. XPath

Query that selects c-nodes that have a b-child?  

//c[b]
XPath

Query that selects c-nodes that have a b-child?  //c[b]

C-nodes that have NO b-child?
XPath

Query that selects c-nodes that have a b-child?  \(/c[b]\)

c-nodes that have NO b-child?  \(/c[not(b)]\)
XPath

Query that selects c-nodes that have a b-child?  //c[b]

c-nodes that do NOT have a b-child?  //c[not(b)]

NEGATE

does there exist a b-node?
XPath

Query that selects c-nodes that have a b-child?  
//c[b]

c-nodes that have a child not labeled b?
XPath

Query that selects c-nodes that have a b-child?  
//c[b]

c-nodes that have a child not labeled b?  
//c[[not(self::b)]]
XPath

Query that selects a-attributes with maximal value?
Query that selects \texttt{a}-attributes with maximal value?

\[
\text{//@a[not(.< //@a)]}
\]
XPath

Query that selects element nodes at level 3 of the tree?
Query that selects element nodes at level 3 of the tree?  /*/*/*/*/*
XPath

Query that selects element and text nodes at level 3 of the tree?
XPath

Query that selects element and text nodes at level 3 of the tree?

/*/*/child::node()

//node()[count(ancestor::*[*])=3]
An XPATH Tester Tool which runs an XPATH statement against an XML fragment

**XPATH Statement**

`//node()[count(ancestor::*>=3]`

**Result**

```
foo
-----
<d/>
------
<d/>
------
foo
-----
bar
```
Useful Functions (on Node Sets)

→ `last()`
  returns context-size from the evaluation context

→ `position()`
  Returns context-position from the eval. context

Recall: //a abbreviates //descendant-or-self::node()/child::a
Useful Functions (on Node Sets)

→ last()
returns context-size from the evaluation context

→ position()
Returns context-position from the eval. context

Recall: //a abbreviates //descendant-or-self::node()/child::a

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

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

→ `last()`
returns context-size from the evaluation context

→ `position()`
Returns context-position from the eval. context

Recall: //a abbreviates //descendant-or-self::node()/child::a
Useful Functions (on Node Sets)

→ last()
returns context-size from the evaluation context

→ position()
Returns context-position from the eval. context

```xml
<a b="1">b</a>
  a="1"
  b="1"
  c "foo"
  d "foo"

<c a="1.0">c</c>
  a="2"
  d "bar"
  d

<d a=""/>"
Useful Functions (on Node Sets)

> last()
returns context-size from the evaluation context

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

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

```
// *[position()=last()-1
and ./text()="foo"]
```

Useful:
```xml
child::*[self::chapter or self::appendix][position()=last()]
selects the last chapter or appendix child of the context node
```
Useful Functions (on Node Sets)

→ last()
returns context-size from the evaluation context

→ position()
Returns context-position from the eval. context

```c
// Example
a = new Node();
b = new Node();
c = new Node();
da = new Node();
b = new Node();
c = new Node();
da = new Node();

// Use
last(); // Context-size
position(); // Context-position
```

→ allows absolute location of any node (a la Dewey)
Useful Functions (on Node Sets)

→ `last()`
  returns context-size from the evaluation context

→ `position()`
  Returns context-position from the eval. context

*a=b="1"
b="1"

*a="1"

*a="2"

*b="1"

*a="1.0"

Abbreviation:

*/*[position()=1]/*[position()=2]/*[position()=2]
Useful Functions (on Node Sets)

→ `last()`
returns context-size from the evaluation context

→ `position()`
Returns context-position from the eval. context

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

Abbreviation:  /*[[1]]//*[[2]]//*[[2]]  ➔ What is result for  /*[[./*[[2]]//*[[2]]]]
Useful Functions (on Node Sets)

→ `last()`
returns context-size from the evaluation context

→ `position()`
Returns context-position from the eval. context

What is the result for
`//d[position() = last()]`

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

Abbreviation: *
```/*[1]/*//*[2]/*//*[2] */What is result for *//*[ ./*[2]/*//*[2] ]
```
Useful Functions (on Node Sets)

→ `last()`
  returns context-size from the evaluation context

→ `position()`
  Returns context-position from the eval. context

How do you select the last 20 book-children of books?
Useful Functions (on Node Sets)

→ `last()`
returns context-size from the evaluation context

→ `position()`
Returns context-position from the eval. context

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

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

→ can be done polynomial time

→ based on “context-value tables”

→ careful recording of unique values (no duplicates)
2. XSLT
2. XSLT

→ Extensible Stylesheet Language Transformations

→ W3C Standard

→ developed early on (XSL in 1998)

→ Version 2.0 (W3C recommendation 2007)

→ Version 3.0 (streaming)
   W3C candidate recommendation (15. November 2015)

All major browsers (Chrome, Firefox, IE, Safari, etc) support XSLT

→ if an XML document has an XSL stylesheet associated with it, the browser will transform it on-the-fly
XML, typical usage scenario

Document structure
- Def. of price, gst, ...
- DTD, XML Schema

```
<Product>
  <product_id> d101 </product_id>
  <name> Sony discman </name>
  <currency> AUD </currency>
  <price> 169.00 </price>
  <gst> 10% </gst>
</Product>
...
```

Presentation
- Format info

XML Stylesheet

One data source → many (dynamic gen.) views
XML

- General data format
- Includes no formatting information

Usage Idea

- Store data once, in most general format
  (free tech writers from bothering with layout issues)

- Reuse fragments of the data (same contents; looks different depending on the context)

E.g. different output formats (size, device)
  style tailored to reader’s preference
  style tailored to adhere w. corporate/web site identity
eXtensible Stylesheet Language

“stylesheet” = recipe how to display your XML data

e.g. “display titles of books in bold, large font”
    “display authors of books in italic, medium size”
    etc.

→ Choose an output format (e.g. XHTML, HTML, text, PDF, etc.)

→ Transform the XML: - add formatting information
    - rearrange data
    - sorting
    - delete certain parts, copy others
    etc
Example Transformations

→ XML to HTML – for browser display
→ XML to LaTeX – for TeX layout
→ XML to SVG – graphs, charts, trees
→ XML to tab-delimited – for DB/stat packages
→ XML to plain text, e.g., PDF – for printing

  e.g. print bills for a telecom company
  → data comes in XML
  → want to produce bills in PDF
Example Transformations

- XML to HTML – for browser display
- XML to LaTeX – for TeX layout
- XML to SVG – graphs, charts, trees
- XML to tab-delimited – for DB/stat packages
- XML to plain text, e.g., PDF – for printing

E.g. print bills for a telecom company
→ data comes in XML
→ want to produce bills in PDF

XSLT vs. custom SAX/DOM traversal (as your XML→CSV from Assignment 1)

→ smaller code
→ better readability & maintainability
→ let XSLT compiler attempt to stream the transformation
Example Transformations

→ smaller code
→ better readability & maintainability
→ let XSLT compiler attempt to *stream* the transformation

→ “best effort” approach
→ very hard to detect statically (interesting research question)
→ XSLT 3.0

Some academic transformation languages even have

→ static type checking
→ compiler can verify if all outputs are, e.g., correct XHTML
XSLT program

XML document

XSLT Implementation

output string

XSLT program: list of template (rules) – written in XML syntax

Template: pattern → action

Template

<xs1:template match="para">
  <p><xs1:apply-templates/></p>
</xs1:template>

(output) XPath expression → current node labeled “para”?

output p-node (in XML)

recursively descend & apply templates
XSLT program
→ changes “para” nodes to “p”-nodes
→ deletes all other element nodes

$ cat t1.xml
<x><t/><para><para/></para></para></x>

$ xsltproc para.xslt t1.xml
<?xml version="1.0"?>
<p><p/></p>
XSLT program
→ changes “para” nodes to “p”-nodes
→ deletes all other element nodes
→ keeps all text-nodes!

$ cat t2.xml
<a z="TT">this<b>is a <d a="1278"/></e/>)</b>
<para u="18">hello world</para>

$ xsltproc para.xslt t2.xml
<?xml version="1.0"?>
this is a <p>hello world</p>
→ all nodes are matched and taken over
→ **except** attributes nodes (they are deleted)
$ cat t2.xml
<a z="TT">this<b>is a <d a="1278"/><e/></b>
<para u="18">hello world<c/></para></a>

$ xsltproc identity.xslt t2.xml
<?xml version="1.0"?>
<a z="TT">this<b>is a <d a="1278"/><e/></b>
<para u="18">hello world<c/></para></a>

→ all nodes are matched and taken over
$ cat t2.xml
<a z="TT">this<b>is a <d a="1278"/><e/></b></a>
<para u="18">hello world<c/></para></a>

$ xsltproc identity.xslt t2.xml
<?xml version="1.0"?>
<a z="TT">this<b>is a <d a="1278"/><e/></b></a>
<para u="18">hello world<c/></para></a>

→ all nodes are matched and taken over
→ compare the (size of the) above XSLT program to a SAX program for the same task!
all nodes are matched and taken over
compare the (size of the) above XSLT program to SAX program for the same task!

would be interesting to rewrite the ebay XML→ CSV converter in XSLT
XPATH Tester/Evaluator/Query

An XPATH Tester Tool which runs an XPATH statement against an XML fragment

**XPATH Statement**

```
//node() | //@*
```

**Run XPATH**

**Result**

```
<root>
  <a b="1">
    <b a="1">
      <d>
        <c a="2"/>
      </d>
    </b>
  </a>
  <foo/>
  <b a="1">
    <d>
      <c a="2"/>
    </d>
  </b>
</root>
```

**XML**

```
<a b="1">b a="1"><d>foo</d><c a="2"><d/></c><b><c a="1.0">b>foo</b><d>bar</d></c></a>
```

What will this program do?
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="node() | @*">
    <xsl:copy>
      <xsl:apply-templates select="."/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>

$ xsltproc identity.xslt t2.xml
runtime error: file loop.xslt line 6 element copy
xsltApplyXSLTTemplate: A potential infinite template recursion was detected. You can adjust xsltMaxDepth (--maxdepth) in order to raise the maximum number of nested template calls and variables/params (currently set to 3000).

Templates:
#0 name node() | @*
#1 name node() | @*
#2 name node() | @*
→ XSLT is Turing-complete
→ easy to write non-terminating programs
→ both templates are applicable to b-nodes
→  both templates are applicable to **b**-nodes
→  **more specific** template is used ("b" before "node()")
Conflict Resolution and Modes in XSLT

- Note that for each node visited by the XSLT processor (cf. default template ②), **more than one template might yield a match**.
- XSLT assigns a **priority** to each template. The more specific the template pattern, the higher the priority:

  ```xml
  <xsl:template match="e"> cons </xsl:template>
  ```

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>−0.5</td>
</tr>
<tr>
<td><code>ns:*</code></td>
<td>−0.25</td>
</tr>
<tr>
<td>element/attribute name</td>
<td>0</td>
</tr>
<tr>
<td>any other XPath expression</td>
<td>0.5</td>
</tr>
</tbody>
</table>

- **Example:**
  Priority of author is 0, priority of `/strip/prolog/author` is 0.5.

- Alternatively, make priority explicit:
  ```xml
  <xsl:template priority="p"> ... </xsl:template>
  ```
Delete all nested <list>s

```xml
<xsl:template match="list/list"
   priority="2">
   <!-- deleted nested list -->
</xsl:template>

<xsl:template match="list"
   priority="1">
   <list><xsl:apply-templates/></list>
</xsl:template>

<d>
<list>text1
<list>
<list>blah
</list>
</list>
<list>next
</list>
</d>

<?xml version="1.0"?>
<list>text1
</list>
</list>
<list>
</list>
</d>
```
Delete all nested `<list>`s

```
<xsl:template match="list/list">
  <!-- deleted nested list -->
</xsl:template>

<xsl:template match="list">
  <list><xsl:apply-templates/></list>
</xsl:template>
```

```
<d>
  <list>text1</list>
  <list>blah</list>
</d>
```

```
<?xml version="1.0"?>
<list>text1
  <list>
    <list>
      next
    </list>
  </list>
</list>
```

→ if no priorities are given, what is the output?
Question

How do you remove `book//author/first` elements, but keep everything else in the XML?
Question

How do you remove `book//author/first` elements, but keep everything else in the XML?

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="book//author/first" priority="5"/>
</xsl:template>

<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```
Context

Quite often, an XSLT stylesheet wants to be context-aware.

- Since the XSLT priority mechanism is not dynamic, this can cause problems.

Example: Transform the following XML document (sectioned text with cross references) into XHTML:

```xml
<section id="intro">
  <title>Introduction</title>
  <para>This section is self-referential: <xref to="intro"/></para>
</section>
```

We want to generate XHTML code that looks somewhat like this:

```html
<h1>Introduction</h1>
<p>This section is self-referential: <em>Introduction</em>.</p>
```

⚠️ The section title needs to be processed twice, once to produce the heading and once to produce the cross reference.
The “obvious” XSLT stylesheet produces erroneous output:

```xml
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">

    <xsl:template match="title">
        <h1><xsl:apply-templates/></h1>
    </xsl:template>

    <xsl:template>
    </xsl:template>

    <xsl:template match="para">
        <p><xsl:apply-templates/></p>
    </xsl:template>

    <xsl:template match="xref">
        <xsl:apply-templates select="id(@to)/title"/>
    </xsl:template>

</xsl:stylesheet>
```

```html
<h1>Introduction</h1>
<p>This section is self-referential: <h1>Introduction</h1>. </p>
```
XSLT modes

- We need to make the processing of the title element aware of the context (or **mode**) it is used in: inside an xref or not.

- This is a job for **XSLT modes**.
  - In `<xsl:apply-templates>` switch to a certain mode `m` depending on the context:
    
    ```xml
    <xsl:apply-templates mode="m" .../>
    ```
  - After mode switching, only `<xsl:template>` instructions with a mode attribute of value `m` will match:
    
    ```xml
    <xsl:template mode="m" .../>
    ```
  - As soon as `<xsl:apply-templates mode="m" .../>` has finished matching nodes, the previous mode (if any) is restored.
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">

<xsl:template match="title">
    <h1><xsl:apply-templates/></h1>
</xsl:template>

<xsl:template match="title" mode="ref">
    <em><xsl:apply-templates/></em>
</xsl:template>

<xsl:template match="xref">
    <xsl:apply-templates select="id(@to)//title" mode="ref"/>
</xsl:template>

</xsl:stylesheet>

<h1>Introduction</h1>
<p>This section is self-referential: <em>Introduction</em>. </p>
### More on XSLT

<table>
<thead>
<tr>
<th>XSLT Instruction</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsl:choose, xsl:when</td>
<td>switch statement (ala C)</td>
</tr>
<tr>
<td>xsl:call-template</td>
<td>explicitly invoke a (named) template</td>
</tr>
<tr>
<td>xsl:for-each</td>
<td>replicate result construction for a sequence of nodes</td>
</tr>
<tr>
<td>xsl:import</td>
<td>import instructions from another stylesheet</td>
</tr>
<tr>
<td>xsl:output</td>
<td>influence XSLT processor’s output behaviour</td>
</tr>
<tr>
<td>xsl:variable</td>
<td>set/read variables</td>
</tr>
</tbody>
</table>

- For a complete XSLT reference, refer to [W3C](http://www.w3.org/TR/xslt)
- Apache’s Cocoon is an XSLT-enabled web server (see [http://xml.apache.org/cocoon/](http://xml.apache.org/cocoon/)).
<xsl:output method="xml|html|text|name"
version="string"
encoding="string"
omit-xml-declaration="yes|no"
standalone="yes|no"
doctype-public="string"
doctype-system="string"
cdata-section-elements="namelist"
indent="yes|no"
media-type="string"/>
This example declares a **global variable** `para-font-size`, which it references in an attribute value template.

```xml
<xsl:variable name="para-font-size">12pt</xsl:variable>

<xsl:template match="para">
  <fo:block font-size="${para-font-size}">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```
<xsl:number>

- Used to generate auto-numbering

  <xsl:number
    level="single|multiple|any"
    count="pattern" -- which nodes count?
    from="pattern" -- starting point
    value="number-expr" -- force value
    format="s" -- (not covering)
    lang="lg" -- lang to use
    letter-value="alphabetic|traditional"
    grouping-separator="char" -- 1,000
    grouping-size="number" -- 3 in EN

/>
<colors>
  <color>red</color>
  <color>green</color>
  <color>blue</color>
  <color>yellow</color>
</colors>

<xsl:template match="color">
  <xsl:number/>.
  <xsl:apply-templates/>
</xsl:template>

1. red
2. green
3. blue
4. yellow
<colors>
  <color>red</color>
  <color>green</color>
  <color>blue
    <color>robin's egg</color>
    <color>navy</color>
    <color>cerulean</color>
  </color>
  <color>yellow</color>
</colors>

<xsl:template match="color">
  <xsl:number level="multiple" format="1. "/>
  <xsl:apply-templates/>
</xsl:template>

1. red
2. green
3. blue
  3.1. robin's egg
  3.2. navy
  3.3. cerulean
4. yellow
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:number level="multiple" count="*"/>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
<?xml version="1.0"?>
<c>
  <c>
    <AA>
      <a1/>
      <a2/>
      <a3><u/></a3>
    </AA>
  </c>
  <b>
    <BB>
      <b1/>
      <b2/>
      <b3/>
    </BB>
  </b>
</c>
That's all folks!

This ends the main lectures of “Applied Databases”.

→ Lectures 19 & 20 (next week): Recap / Exam preparation

→ no lectures after that!  [ i.e., no lectures on April 3 & 6 ]

Thank you for your attention!