5.1 Additional features of XPath & XSLT

- XPath support for
  - arithmetics
  - processing ID/IDREF cross-references
  - manipulation of strings
- Generating text
  - for content
  - for attribute values
- Repetition, sorting and conditional processing
- Generating numbers

XPath: Arithmetical Operations

- Operators for double-precision (64 bit) floating-point arithmetics
  - +, -, *, div, mod (same as % in Java)
- Mapping numbers to integers:
  - floor(x) = max{nz| n ≤ x}
  - ceiling(x) = min{nz| n ≥ x}
  - round(x) = floor(x+0.5)
- Formatting numbers as strings (example):
  - format-number(-1.2534, "0.0") = "-1.3"

Aggregate Functions

- Counting nodes
  - count(node-set)
  - and summing them as numbers
    - sum(node-set)
- Example:
  - Grade average for the first student in document:
    - sum({//student}[1]/course[@grade]) div count({//student}[1]/course)

Cross-referencing

- Function id selects elements by their unique ID
  - NB: ID attributes need to be declared in DTD
  - See an example later
- Examples:
  - id(sect:intro)
  - selects the element with unique ID "sect:intro"
  - id(sect:intro)/para[5]
  - selects the fifth para child of the above element
  - id(sect1 sect2 sect3)
  - selects 3 sections (if they have the corresponding ID values)

String manipulation

- Equality and inequality of strings can be tested with operators = and !=
  - "foo" = "foo"; "foo" != "Foo"
- Testing for substrings:
  - starts-with("dogbert", "dog") = true()
  - contains("dogbert", "gbe") = true()
- Concatenation (of two or more strings),
  - concat("dog", "bert") = "dogbert"

XPath: more string functions

- substring-before("ftp://a", "/") = substring-before("ftp://a", "/") = "ftp:
- substring-after("ftp://a", "/") = "/a"
- substring( string, start, length? )
  - substring("dogbert", 1, 3) = "dog"
  - substring("dogbert", 3) = "gbe"
  - string-length("dogbert") = 7
- translate( Str, Replaced, Replacing )
  - translate("doggy", "dgo", "Sss") = "Sissy"

Generating Text

- The string-value of an expression can be inserted in the result tree by instruction
  - <xsl:value-of select="Expr" />
  - If Expr evaluates to a node-set, value of the first node in document order is used
- Consider transforming source elements like
  - <name alias="Bird"> <first>Charlie</first> <last>Parker</last> </name>
    - to the form
      - Charlie ("Bird") Parker

Computing generated text (2)

- This can be specified by template rule
  - <xsl:template match="name">
    - <xsl:value-of select="first" />
      - ("<xsl:value-of select="@alias" />")
    - <xsl:value-of select="last" />
    - </xsl:text>
  - </xsl:template>
- Verbatim text (like the white-space above) can be inserted using xsl:text
**XSLT: Repetition**

- Nodes can be "pulled" from source for processing using
  
  ```xml
  <xsl:for-each select="Expr">
    Template
  </xsl:for-each>
  ```

- The template is applied to each of the selected nodes (0, 1 or more), each node in turn as the current() node
  
  - in document order, unless sorted using `xsl:sort` instructions (see later)

**Example: Table of contents**

- A table of contents can be formed of section titles:
  
  ```xml
  <xsl:template match="/">
    <HTML><HEAD><TITLE>$xsl:value-of select="document/title" /></TITLE></HEAD>
    <BODY>
      <H2>Table of Contents</H2>
      <OL>
        <xsl:for-each select="//section/title">
          <LI><xsl:value-of select=". "/></LI>
        </xsl:for-each>
      </OL>
    </BODY>
  </xsl:template>
  ```

**Example (of xsl:for-each)**

- Consider formatting the below document as HTML:
  
  ```xml
  <DOCTYPE document= "ATTILUST section is ID IMPLIED">
  <document>  <title>The Joy of XML</title>
    <section id= "Intro">Getting Started</section>
    <section id= "Intro">Family affairs</section>
    <section id= "Intro">Finishing Up</section>
  </document>
  ```

- All names can be collected in a last-name-first-name order using the below template
  
  ```xml
  <H2>Family affairs</H2>
  <UL>
    <xsl:for-each select="//name">
      <xsl:value-of select="concat("first"," ","last")" />
    </xsl:for-each>
  </UL>
  ```

- With this rule the source fragment
  
  ```xml
  <name><first>Dave</first><last>Dobrik</last></name>
  <name><first>Helen</first></name>
  <name><first>Bob</first></name>
  ```

  becomes
  
  ```xml
  <name>Dave Dobrik</name>
  <name>Helen</name>
  <name>Bob</name>
  ```

  As we discussed in `<title-ref idref="Intro" />`, processing XML documents is fun.

**Example (cont; Cross references)**

- Cross references (to sections) can also be processed using `xsl:for-each`:
  
  ```xml
  <xsl:for-each select="//idref">
    <xsl:value-of select="/section[position()=. and name()=.]"></xsl:value-of>
  </xsl:for-each>
  ```

- All names can be collected in a last-name-first-name order using the below template
  
  ```xml
  <H2>Index</H2>
  <UL>
    <xsl:apply-templates select="document/section"/>
  </UL>
  ```

- This creates an UL list with items
  
  ```xml
  <UL>
    <LI>Brown, Helen</LI>
    <LI>Brown, Dave</LI>
    <LI>Brown, Bob</LI>
  </UL>
  ```
What about duplicates?

- Is it possible to eliminate duplicate values like
  `<LI>Brown, Helen</LI>  
  `<LI>Brown, Helen</LI>`?
- Yes (but not that straightforward)
- Using conditional instructions
  - See next

Conditional processing

- A template can be instantiated or ignored based on the value of a test: Boolean expression, using
  `<xsl:if test="Expression">  
    Template</xsl:if>`
- Example: a comma-separated list of names:
  `<xsl:template match="namelist/name">  
    <xsl:if test="position() &lt; last()">  
      <xsl:apply-templates/>
    </xsl:if></xsl:template>`

Generating Numbers

- Formatted numbers can be inserted in the result tree by element `xsl:number`
  - number can be specified by attribute `value="Expr"`
  - otherwise the number generated by the position of the current node in the source tree
- We'll consider typical cases through examples
  - The complete rules of the Spec are rather complex
- Example 1: Numbering list items
  ```xml
  <xsl:choose>  
    <xsl:when test="Expr"> ... </xsl:when>  
    ...  
    <xsl:otherwise> ... </xsl:otherwise>
  </xsl:choose>
  ```

Generating numbers: Example 1

```xml
<xsl:template match="ol/item">
  <!-- default: count similar siblings (items) -->  
  <xsl:variable name="count" select="count(preceding::item)"/>
  <xsl:apply-templates/>
</xsl:template>
```

Generating numbers: Example 2

- Hierarchical numbering (1, 1.1, 1.1.1, 1.1.2, ...) for titles of chapters, titles of their sections, and titles of subsections:
  ```xml
  <xsl:template match="title">
    <xsl:number level="multiple"  
      count="chapter|section|subsection"  
      format="1.1 "/>
  </xsl:template>
  ```
Example 2: Variation

As above, but number titles within appendices with A, A.1, A.1.1, B.1 etc:

```xml
<xsl:template match="appendix/title">
  <xsl:number level="multiple" count="appendix|sect|subsect" format="A.1 "/>
  <xsl:apply-templates/>
</xsl:template>
```

Generating numbers: Example 3

Sequential numbering of notes within chapters:
(more precisely: starting anew at the start of any chapter)

```xml
<xsl:template match="note">
  <xsl:number level="any" from="chap" format="(1) "/>
  <xsl:apply-templates/>
</xsl:template>
```

Ex 3: Sequential numbering from chaps

```
(1) Yes!
(2) No!
(3) Perhaps?
(4) OK
```