5.1 Additional features of XPath & XSLT

- XPath support for:
  - arithmetical operations
  - 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

- Operations for double-precision (64 bit) floating-point numbers
  - +, -, *, div, mod (same as % in Java)
  - functions to map numbers to integers:
    - floor(-1.1) = -2, floor(1.1)=floor(1.5)=1
    - ceiling(-1.1) = -1, ceiling(1.1)=ceiling(1.5)=2
    - round(-1.1) = -1, round(1.1)= 1, round(-1.5) = -1, round(1.5) = 2

Cross-referencing

- Function `id` selects elements by their unique ID
  - NB: ID attributes need to be declared (in DTD or its internal subset; 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"
- Concatenation (of two or more strings),
  - `concat("dog", "bert") = "dogbert"
- Testing for substrings:
  - `starts-with("dogbert", "dog") = true`
  - `contains("dogbert", "gbe") = true`

XPath: more string functions

- `substring-before("dogbert", "bert") = "dog"
- substring-before("dogbert", "h") = "dog"
- substring-after("dogbert", "g") = "bert"
- `substring(\"dogbert\", 1, 3) = "dog"
- substring("dogbert", 3) = "gbert"
- string-length("dogbert") = 7
- `translate(\"Str\", \"ReplacedChars\", \"ReplacingChars\")`:
  - `translate(\"Dogbert\", \"dog\", \"Bil\") = "Bilbert"

Computing generated text

- The string value of an expression can be inserted in the result tree by instruction:
  `<xsl:value-of select="Expr" />`
  - if the expression evaluates to a `node-set` the 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
```
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:template>
```

Verbatim text (like the white space above) can be inserted using `xsl:text`

```
<xsl:text>
</xsl:text>
```

The string-value of an expression can be inserted in an attribute value by surrounding the expression by braces `{}` and `}`

```
Consider transforming source element

```
```

Expressions `{file}` and `{size/@width}` are evaluated in the context of the current node (the `photo` element)

```
```

Nodes can be "pulled" from source for processing using instruction

```
<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 (of `xsl:for-each`)

```
Consider formatting the below document as HTML:

```
```

A table of contents can be formed of section titles:

```
```

Example: Table of contents

```
```
Cross references (to sections) can also be processed using 
\texttt{xsl:for-each:}
\begin{verbatim}
<xsl:template match="title-ref">
  <xsl:for-each select="id[(@idref)]">
    Section (\texttt{value-of select="substring(title, 1, 8)"}) ...
  </xsl:for-each>
</xsl:template>
\end{verbatim}

With this rule the source fragment
\begin{verbatim}
As we discussed in <title-ref idref="Intro"/>
\end{verbatim}
becomes
\begin{verbatim}
As we discussed in Section (Getting ...)
\end{verbatim}

A sorted order for the processing of nodes with 
\texttt{xsl:for-each} and \texttt{xsl:apply-templates} can be specified by \texttt{xsl:sort:}

controlled by attributes of \texttt{xsl:sort} like

- \texttt{select}: expression for the sort key
- \texttt{data-type}: "text" (default) or "number"
- \texttt{order}: "ascending" (default) or "descending"

The first \texttt{xsl:sort} specifies the primary sort key, the second one the secondary sort key, and so on.

All names can be collected in a last-name-first-name order using the below template
\begin{verbatim}
<xsl:choose>
  <xsl:when test="Expr1"> ... </xsl:when>
  <xsl:when test="Expr2"> ... </xsl:when>
  ... 
  <xsl:otherwise> ... </xsl:otherwise>
</xsl:choose>
\end{verbatim}

Also a case-like construct:
\begin{verbatim}
<xsl:choose>
  <!-- The first 'when' whose test=true() is instantiated: -->
  <xsl:when test="Expr"> ... </xsl:when>
  <xsl:when test="Expr2"> ... </xsl:when>
  ... 
  <!-- If no 'when' applies, optional 'otherwise' is instantiated: -->
  <xsl:otherwise> ... </xsl:otherwise>
</xsl:choose>
\end{verbatim}
Example (cont; Eliminating duplicate names)

- No access to other nodes (except current()) in the list of xsl:for-each (?)
  - But can refer to other nodes in the source tree
  - Process just the first one of duplicate names:
    - xsl:sort select="/name"
    - xsl:sort select="/first" /n
    - xsl:if test="not(
      preceding::name[first=current()/first and last=current()/last] )">
      <LI><xsl:value-of select="last" />, <xsl:value-of select="first"/></LI>
    </xsl:if>
  </xsl:for-each>

Generating numbers (1/3)

- 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 based on the position of the current node in the source tree

Example 1, a numbered list:

```xml
<xsl:template match="/ol/item">
  <!-- default: count like siblings (items) -->
  <xsl:apply-templates/>
</xsl:template>
```

Generating numbers (2/3)

- Example 2:
  Hierarchical numbering (1, 1.1, 1.1.1, 1.1.2, ...) for titles of chapters, titles of sections of chapters,
  and titles of subsections of sections

```xml
<xsl:template match="/title">
  <xsl:number level="multiple"
    count="chapter|section|subsection" format="1.1 "/>
  <xsl:apply-templates/>
</xsl:template>
```

Generating numbers (3/3)

- Example 3:
  Sequential numbering of notes within chapters:

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