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{n \in \mathbb{Z} : n \leq x}
  » ceiling(x) = min{n \in \mathbb{Z} : n \geq x}
  » round(x) = floor(x+0.5)
- Mapping numbers to formatted 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:
  - Course grade average for the first student:
    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 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"
- Testing for substrings:
  - starts-with("dogbert", "dog") = true()
  - contains("dogbert", "gb") = true()
- Concatenation (of two or more strings),
  - concat("dog", "bert") = "dogbert"

XPath: more string functions

- substring-before("dogbert", "bert") = substring-before("dogbert", "b") = "dog"
- substring-after("dogbert", "g") = "bert"
- substring(string, startpos, length?)
  » substring("dogbert", 1, 3) = "dog"
  » substring("dogbert", 3) = "gbert"
- string-length("dogbert")=7
- translate(Str, ReplacedChars, ReplacingChars):
  » translate("dogbert", "dg", "Di") = "Dilbert"

Generating Text

- The string-value of an expression can be inserted in the result tree by instruction
  `<xsl:value-of select="Expr" />
  - if Expr selects more than one nodes, 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>`
  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
Attribute value templates

- The string-value of an expression can be inserted in an attribute value by surrounding the expression by braces { and }
- Consider transforming source element

```xml
<br><file>Mary.jpg</file>
<size width="300"/>
</br>
```

into form

```xml
<img src="/images/Mary.jpg" width="300" /> 
```

Attribute value templates (2)

- This can be specified by template rule

```xml
<xsl:template match="photo">
  <img src="/images/{file}" width="{size/@width}" /> 
</xsl:template>
```

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

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

```xml
  » 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:apply-templates /></LI>
  </xsl:for-each>
  </OL>
  </BODY> </HTML>
</xsl:template>
```

Example (of xsl:for-each)

- Consider formatting the below document as HTML:

```xml
<!DOCTYPE document [ <!ATTLIST section id ID #IMPLIED> ]>
<document>
<title>The Joy of XML</title>
<section id="Intro"><title>Getting Started</title>

```xml
  <name><first>Helen</first> <last>Brown</last></name>
  says that processing XML documents is fun.
  <name><first>Dave</first> <last>Dobrik</last></name> agrees.
</section>

<section>

```xml
  <name><first>Bob</first> <last>Brown</last></name> is the husband of
  <name><first>Helen</first> <last>Brown</last></name>.
</section>

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

Example: Sorted index of names

- All names can be collected in a last-name-first-name order using the below template

```xml
<xsl:template match="/">
  <H2>Index</H2> <UL>
  <xsl:for-each select="/name">
  <xsl:sort select="last" /> 
  <xsl:sort select="first" /> 
  <LI><xsl:value-of select="last" />, <xsl:value-of select="first" /></LI>
  </xsl:for-each>
  </UL> 
</xsl:template>
```

XSLT Sorting

- A sorted order for the processing of nodes with xsl:for-each and xsl:apply-templates can be specified by xsl:sort
- controlled by attributes of xsl:sort like:
  - select: expression for the sort key
  - data-type: "text" (default) or "number"
  - order: "ascending" (default) or "descending"
- The first xsl:sort specifies the primary sort key, the second one the secondary sort key, and so on.

Example (cont; Sorted index of names)

- All names can be collected in a last-name-first-name order using the below template

```xml
<xsl:for-each select="/name">
  <xsl:sort select="name" data-type="text" order="ascending"/>
  <LI><xsl:value-of select="name" /></LI>
</xsl:for-each>
</UL>
```

- This creates an UL list with items

```xml
  <LI>Brown, Bob</LI>
  <LI>Brown, Helen</LI>
  <LI>Dobrik, Dave</LI>
```
What about duplicates?

- Is it possible to eliminate duplicate values like `<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:for-each select="namelist/name"> ...
  <xsl:if test="position() &lt; last()"> , </xsl:if>
  </xsl:for-each>`

Conditional processing (2)

- Also a case-like construct (~ switch in Java):
  `<xsl:choose>
  <!-- The first 'when' whose test=true() is instantiated: -->
  <xsl:when test="Expr"></xsl:when>
  <xsl:when test="Expr"></xsl:when>
  ...
  <!-- If no 'when' applies, an optional 'otherwise' is instantiated: -->
  <xsl:otherwise> ... </xsl:otherwise>
  </xsl:choose>

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 based on 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

Generating numbers: Example 1

```xml
<xsl:template match="ol/item">
  <!-- default: count like siblings (items) -->
  <xsl:number format="1. "/>
  <xsl:apply-templates/>
</xsl:template>
```

Generating numbers: Example 2

```xml
<xsl:template match="ol/item">
  <!-- default: count like siblings (items) -->
  <xsl:number level="multiple" count="chap|sect|subsect" format="1.1 "/>
  <xsl:apply-templates/>
</xsl:template>
```

Generating numbers: Example 2

```xml
<xsl:template match="ol/item">
  <!-- default: count like siblings (items) -->
  <xsl:number level="multiple" count="chap|sect|subsect" format="1.1.1 "/>
  <xsl:apply-templates/>
</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:number>
</xsl:template>
```

Generating numbers: Example 3

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

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