5.1 Additional Features

- XPath for arithmetics, cross-references, and string manipulation
- Generating text
  - for content
  - for attribute values
- Repetition, sorting and conditional processing
- Numbering document contents

5.2 Computing with XSLT

### XPath: Arithmetics

- Double-precision floating-point operators
  
  +, -, *, div, mod (same as in Java)
  
  - e.g. 2.3 mod 1.1 = 0.1
- Rounding numbers up, down, and to the closest integer:
  
  floor(x), ceiling(x), round(x)
- Formatting numbers as strings (e.g.):
  
  Number format: format-number('-1.2534', '0.0') = ' -1.3'
- XSLT 1.0 function; uses Java decimal format patterns

### Aggregate Functions

- Counting nodes
  
  » count (node-set)
  
  - and summing them as numbers
  
  » sum (node-set)
- Example:
  
  - Average of observed temps below current node:
    
    sum(../obs/temperature) / count(../obs)

### Cross-referencing

- Function id selects elements by their unique ID
  
  - NB: ID attributes must be declared in DTD
    
    (See an example later)
- Examples:
  
  - id(sect:intro) selects the element with unique ID "sect:intro"
  - id(sect:intro)/auth[3] selects the third auth of the above element
  - id('sect1 sect2 sect3') selects 3 sections (with corresponding ID values)

### String manipulation

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

### Computing generated text (2)

- This can be specified by template rule
  
  ```xml
  <xsl:template match="name">
    <xsl:value-of select="first" />
    
    ("<xsl:value-of select="@alias" />")
    
    <xsl:value-of select="last" />
  </xsl:template>
  </xsl:template>
  </xsl:stylesheet>
  ```

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

---

**Computing generated text (2)**

- This can be specified by template rule
  
  ```xml
  <xsl:template match="name">
    <xsl:value-of select="first" />
    
    ("<xsl:value-of select="@alias" />")
    
    <xsl:value-of select="last" />
  </xsl:template>
  </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 })
- Example: Transform source element
  ```xml
  <photo>
  <file>Mary.jpg</file>
  <size width="300"/>
  </photo>
  ```
  into form
  ```xml
  <img src="/images/Mary.jpg" width="300"/>
  ```

**Example: Transform source element into form**

**XSLT: Repetition**

- Nodes can be "pulled" from source for processing using
  ```xml
  <xsl:for-each select="Expr"/>
  ```
- Template is applied to the selected nodelist, each node in turn as the current() node
  - in document order, unless sorted using xsl:sort instructions (see later)
  ```xml
  <MEDIA>
  <image width="300" src="Mary.jpg"/>
  </MEDIA>
  ```

**Example: Table of contents**

- A table of contents can be formed of section titles:
  ```xml
  <xsl:template match="//section/title">
  <LI><xsl:value-of select="document/title"/></LI>
</xsl:template>
```
- Possible to eliminate duplicates?

**XSLT Sorting**

- A sorted order for the processing of nodes with xsl:for-each and xsl:apply-templates can be specified by
  ```xml
  <xsl:sort/>
  ```
- controlled by attributes of xsl:sort, like
  - select: expression for the sort key (default: ".")
  - 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; Cross references)**

- Cross-ref can also be processed for each:
  ```xml
  <xsl:for-each select="//idref">
  <!-- just one selected -->
  <xsl:template>
  <LI>
  <xsl:value-of select="idref/text()"/>
  </LI>
  </xsl:template>
  </xsl:for-each>
  ```
- Possible to eliminate duplicates?

**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="last"/>
  </xsl:for-each>
  ```
- This creates an UL list with items
  ```xml
  <LI>Brown, Bob(1)</LI>
  <LI>Brown, Helen(1)</LI>
  <LI>Dobrik, Dave(1)</LI>
  ```
Conditional processing

- A template can be instantiated or ignored with
  \[
  \text{<xsl:if test="BooleanExpr">}
  \text{Template}
  \text{</xsl:if>}
  \]
- Example: a comma-separated list of names:
  \[
  \text{<xsl:template match="/namelist"/>}
  \text{<xsl:apply-templates/>}
  \text{<xsl:if test="position() &lt; last()"/>}
  \text{</xsl:if>}
  \text{</xsl:template>}

Conditional processing (2)

- A case construct (~ switch in Java):
  \[
  \text{<xsl:choose>}
  \text{<xsl:when test="Expr,"..."/>}
  \text{<xsl:when test="Expr","..."/>}
  \text{...}
  \text{<xsl:otherwise>...</xsl:otherwise>}
  \text{</xsl:choose>}

Numbering Document Contents

- Formatted numbers can be inserted in the result tree by element
  \[
  \text{<xsl:number />}
  \]
  - by the position of the current node in the source tree
  - nodes to be counted specified by a count pattern
  - supports common numbering schemes: single-level, hierarchical, and sequential through levels
- Typical cases in following examples
  - (Complete specification rather complex)
- Example 1: Numbering list items

Generating numbers: Example 1

- <xsl:template match="/item">
  \[
  \text{<xsl:if test="names(i)">}
  \text{<xsl:for-each select="name"/>}
  \text{<xsl:value-of select="name[1]"/>}
  \text{<xsl:sort select="last"/>}
  \text{<xsl:if test="not(
    proceeding::name[1]=current()/first
    and last=current()/last
  )">}
  \text{<xsl:value-of select="name[1]"/>}
  \text{</xsl:if>}
  \text{</xsl:for-each>}
  \]

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:
  \[
  \text{<xsl:template match="/title">}
  \text{<xsl:number level="multiple">
    \text{count="chap | sect | subsect"}
    \text{format="1.1."/>}
  \text{</xsl:number>}
  \text{</xsl:template>}

An aside: Meaning of position()

- Evaluation w.r.t the current node list. The above applied to
  \[
  \text{<namelist><name>/name</name><name>/name</name><namelist>}
  \]
- Only the current() node accessible in current node list
- Process just the first one of duplicate names:
  \[
  \text{<xsl:for-each select="/name"/>}
  \text{<xsl:sort select="first"/>}
  \text{<xsl:if test="not(
    proceeding::name[1]=current()/first
    and last=current()/last
  )">}
  \text{<xsl:value-of select="name[1]"/>}
  \text{</xsl:if>}
  \text{</xsl:for-each>}

Example (cont; Eliminating duplicate names)

- Only the current() node accessible in current node list
- Process just the first one of duplicate names:...
What is the exact computational power of XSLT?
Could we use it for general computing?

XSLT 2.0 iteration over arbitrary sequences
callable variables

Generating numbers: Example 3

Results tree fragments

Computing with XSLT

Further programming-like features:
- **variables** (names bound to non-updatable values):
  ```xsl:variable```
- **callable** named templates with parameters:
  ```xsl:call-template```

Result tree fragments

Visibility of Variable Bindings

The binding is **visible** in following siblings of ```xsl:variable```, and in their descendants:
A Real-Life Example

- We used LaTeX to format an XML article. For this, we needed to map source table structures
  \begin{tabular}{l|c|c|c}
  & & & \\
  \hline
  \hline
  & & & \\
  \hline

to corresponding LaTeX environments:
\begin{tabular}{|l|l|l|l|l|}
\hline
\hline
& \multicolumn{4}{c|}{left-justified cols} \\
\hline
\hline
\end{tabular}

- How to do this?

SDPL 2007 XSLT: Additional Features and Computing 33

More General Solution (1/2)

- Pass the column-count to a named template which generates the requested number of `|`:'s:

```xml
<xsl:template match="tgroup">
  \begin{tabular}{l|c|c|c|c} 
  & & & & \\
  \hline
  \hline
  & & & & \\
  \hline
  \end{tabular}
</xsl:template>
```

SDPL 2007 XSLT: Additional Features and Computing 34

Stylesheet Parameters

- Stylesheet can get params from command line, or through JAXP with

```java
Transformer.setParameter(name, value);
```

SDPL 2007 XSLT: Additional Features and Computing 35

Computational power of XSLT

- XSLT seems quite powerful, but how powerful is it?
  - Implementations provide extension mechanisms, e.g., to call arbitrary Java methods
  - Are there limits to XSLT processing that we can do without extensions?
- Any algorithm can be shown computable with plain XSLT
  - by simulating Turing machines, by a recursive named template with string parameters
  (see 2005 lecture notes, or References)

SDPL 2007 XSLT: Additional Features and Computing 36

What does this mean?

- XSLT has full algorithmic power
  - (It is "Turing-complete")
  - Is this intentional?
    - Awkward as a general-purpose programming language!
    - Impossible to recognise non-terminating transformations automatically
      (the "halting problem" has no algorithmic solution)
      - could attempt "denial-of-service" attacks with non-terminating style sheets(!)

SDPL 2007 XSLT: Additional Features and Computing 37