Chapter 8: XSLT Extensible Stylesheet Language/Transformations

References:

- James Clark (Editor): XSL Transformations (XSLT), Version 1.0
  W3C Recommendation, 16 November 1999
  [https://www.w3.org/TR/xslt]
- Michael Kay (Editor): XSL Transformations (XSLT), Version 2.0
  W3C Recommendation, 23 January 2007
  [http://www.w3.org/TR/xslt20/]
- Michael Kay (Editor): XSL Transformations (XSLT), Version 3.0
  W3C Candidate Recommendation, 19 November 2015
  [http://www.w3.org/TR/xslt-30/]
- Michael Kay: XSLT 2.0 and XPath 2.0 Programmer's Reference (Programmer to Programmer) Wiley, 4th Ed. (June 3, 2008), ISBN-10: 0470192747, 1376 pages.
- Wikipedia (English): XSLT
  [https://en.wikipedia.org/wiki/XSLT]
- Robert Tolksdorf: Vorlesung XML-Technologien (Web Data and Interoperability), Kapitel 6: XSLT: Transformation von XML-Dokumenten.
  Freie Universität Berlin, AG Netzbasierte Informationssysteme, 2015.
  [http://blog.ag-nbi.de/wp-content/uploads/2015/05/06_XSLT.pdf]
- w3schools: XSLT ELement Reference.
  [http://www.w3schools.com/xml/xsl_elementref.asp]

Objectives

After completing this chapter, you should be able to:

- write transformations from XML to XML, or from XML to HTML as an XSLT stylesheet.

  This chapter also explains how a transformation from XML to LaTeX is done with XSLT.

- read and understand given XSLT stylesheets.

Overview

1. Introduction
2. Example XSLT Stylesheet
3. Template Rules: Details
4. Restrictions in XPath 1.0
5. More XSLT Constructs

Introduction (1)

- XML is by itself only a data format:
  - ◇ It contains the data (content), but
  - ◇ does not specify how the elements should be printed or displayed in a browser or on paper.
- The output format is specified with style sheets:
  - ◇ Using Cascading Stylesheets (CSS).
  - ◇ Using XSLT to translate XML to HTML.

    The HTML is then typically formatted with CSS.
  - ◇ Using XSLT to translate XML to XSL-FO.

    For paper/PDF. One can also translate to LaTeX with XSLT.

Introduction (2)

- Many browsers support CSS, which is normally used for HTML web pages, also for XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css"
                 href="mystyle.css"?>
<GRADES-DB>
...
```

- However, this has many restrictions:
  - ◇ With CSS, the elements are formatted in the order in which they are written,
  - ◇ and there is only very limited filtering.

Introduction (3)

- The Extensible Stylesheet Language (XSL) consists of two parts:
  - ◇ XSLT (XSL Transformations) is a mechanism to transform XML documents into XML documents (e.g., with other elements/tags).

    As explained below, the output is not necessarily XML. Even binary files can be generated.
  - ◇ XSL-FO (XSL Formatting Objects) is a set of element types/tags with a specified semantics for displaying them.

    "an XML vocabulary for specifying formatting semantics" [https://www.w3.org/Style/XSL/]

Introduction (4)

- So the idea is to
  - ◇ use XSLT to transform a custom XML file to XSL-FO,
  - ◇ which is then displayed on screen or printed on paper.
- XSL-FO especially supports high-quality printout on paper (or as a PDF file).

  Thus, e.g. splitting a document into pages is important for XSL-FO, whereas it is not important for displaying a web page in a browser. Also, hyphenation is treated. Where possible, properties from CSS2 where taken, and somtimes extended or split into several properties.

Introduction (5)

- XSL has its roots in DSSSL, the Document Style Semantics and Specification Language (for SGML).
- XSLT 1.0 became a W3C recommendation (official standard) on November 16, 1999.

  See [https://www.w3.org/TR/xslt]. The current version is XSLT 2.0 from Januar 23, 2007. [https://www.w3.org/TR/xslt20/].
- XSL 1.0 (which specifies XSL-FO) became a W3C recommendation on October 15, 2001.

  See [https://www.w3.org/TR/2001/REC-xsl-20011015/]
  Current ver.: XSL 1.1 (Dec. 5, 2006) [https://www.w3.org/TR/xsl/]
  Draft: XSL 2.0 (Jan. 17, 2012) [https://www.w3.org/TR/xslfo20/]

Introduction (6)

- Quite often, XSLT is used without XSL-FO:
  - ◇ For instance, XML is transformed to HTML to be displayed in a browser.
  - ◇ Or XSLT is used to transform a given XML document into a differently structured XML document (with different element types/tags).

    In this way, one can adapt an XML file from a business partner to one's own XML structure. Or one can integrate XML files from different sources to a common XML vocabulary.

Introduction (7)

- For translating XML to HTML, XSLT can be used in two places:
  - ◇ Client: the web browser does the mapping,
  - ◇ Server: one uses an XSLT processor to translate XML to HTML, and publishes the HTML files.

    Maybe in addition to the XML files. It is also possible that the HTTP server does the translation on demand: The web browser sends in the HTTP request a list of mime types it understands.
- It seems that browsers today still understand only XSLT 1.0 (which is based on XPath 1.0).

  E.g. documentation of XSLT support in Mozilla Firefox: [https://developer.mozilla.org/en-US/docs/Web/XSLT]

Introduction (8)

- Doing the XML to HTML mapping on Client or Server, continued:
  - ◇ If one does the translation in an intranet only for the employees of the company, one can at least rely on the knowledge which browser is used.
  - ◇ On the global internet, it might be that potential customers use old browsers which do not support XSLT or support it in incompatible ways.

    One can still put the XML file on the server in addition to the HTML file, in order to support semantic web applications (like price comparision services).

XSLT Implementations

- Saxon (from Michael Kay)

  M. Kay is editor of the XSLT 2.0 Spec. Basic version (without static type checking and XQuery→Java compiler) is open source. Supports XSLT 2.0, XPath 2.0, XQuery 1.0. [http://saxon.sourceforge.net/]
- Xalan (Apache) (Java and C++ versions)

  [http://xalan.apache.org/]
  This is mainly a library, but it also has a command line untility.
- `xsltproc/libxslt`

  [http://xmlsoft.org/], [http://xmlsoft.org/XSLT/xsltproc.html]
- AltovaXML Community Edition

  [http://www.softpedia.com/get/Internet/Other-Internet-Related/AltovaXML.shtml]

Overview

Example XML File (1)

- Consider the grades DB with data in attributes:

```
<?xml version='1.0' encoding='UTF-8'?>
<?xml-stylesheet type='text/xsl'
                 href='mystyle.xsl'?>
<GRADES-DB>
    <STUDENT SID='101'
             FIRST='Ann' LAST='Smith'
             EMAIL='smith@acm.org'/>
    <STUDENT SID='102'
             FIRST='Michael' LAST='Jones'/>
    ...
```


Example XML File (2)

- Grades DB (with data in attributes), continued:

```
        <EXERCISE CAT='H' ENO='1'
                  TOPIC='Relational Algebra'
                  MAXPT='10'/>
        ...
        <RESULT SID='101' CAT='H' ENO='1'
                POINTS='10'/>
        ...
    </GRADES-DB>
```

Note: If there is a typing error in the name of the stylesheet, many browsers (e.g., Firefox 43) silently apply the built-in templates (see slide 8-56), which means that the output will be empty if the data is stored in attributes. Textual element content is shown.

Example XSLT Stylesheet (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:html="http://www.w3.org/1999/xhtml"
    xmlns="http://www.w3.org/1999/xhtml"
    exclude-result-prefixes="html">
```

- XSLT stylesheets are written in XML syntax, using the outermost element `stylesheet`.

`transform` is allowed as a synonym. The version number is mandatory.

Example XSLT Stylesheet (2)

- The namespace URI for XSLT elements must be

  `http://www.w3.org/1999/XSL/Transform`.

- In the example, a namespace for XHTML is declared in addition to the namespace for XSLT, and this is also the default namespace.

  So one can write XHTML tags without namespace prefix.

- With `exclude-result-prefixes`, it is specified that in the output of the transformation, the namespace prefix of XHTML tags should not be written.

Example XSLT Stylesheet (3)

```
<xsl:output
    method="xml"
    doctype-system=
      "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"
    doctype-public="-//W3C//DTD XHTML 1.1//EN" />
```

- This specifies how the resulting XDM tree should be printed/serialized (in this case, as XHTML).

  Alternative (classical HTML):

  ```
  <xsl:output method="html"
      encoding="ISO-8859-1"
      doctype-public="-//W3C//DTD HTML 3.2 Final//EN"
      indent="yes" />
  See: [https://www.w3.org/TR/xslt#output]
  ```


Example XSLT Stylesheet (4)

```
<xsl:template match="/">
    <html>
        <head><title>Students</title></head>
        <body>
            <h1>Student List</h1>
            <ul>
                <xsl:apply-templates
                     select="/GRADES-DB/STUDENT"/>
            </ul>
        </body>
    </html>
</xsl:template>
```


Example XSLT Stylesheet (5)

- An XSLT stylesheet is mainly a set of transformation rules called "templates" or "template rules".

  [https://www.w3.org/TR/xslt#rules]

- Each template describes a transformation from a subtree of the input (i.e. a start node and all its descendants) into a tree or list of trees.
- The output of the transformation for a given XML document is given by the rule for the root node "/" of the input tree.

  All other templates are used only if they are called (maybe indirectly) from the pattern for this root node "/" with "`apply-templates`".

Example XSLT Stylesheet (6)

- Each transformation rule (template) consists mainly of two parts:
  - ◇ The attribute "`match`" defines, for which nodes this transformation rule is applicable.

    It is a restricted XPath-expression (called a pattern). [http://www.w3.org/TR/xslt#patterns]
  - ◇ The contents of "`xsl:template`" is a pattern for the output. It is mainly copied to the output tree, but contained XSLT elements are evaluated.

    In the example, the contents contains "`xsl:apply-templates`". Another typical tag used in the contents is "`xsl:value-of`".

Example XSLT Stylesheet (7)

- "`xsl:apply-templates`" will be replaced by the result of applying the transformation recursively to the node which is specified in the "`select`"-attribute.

  The contents of this attribute is an XPath expression. [https://www.w3.org/TR/xslt#section-Applying-Template-Rules]
- If it selects several nodes, the transformation results for all these the nodes are inserted into the output tree (in the same sequence).
- If the attribute "`select`" is omitted, all child nodes are transformed.

Example XSLT Stylesheet (8)

```
    <xsl:template match="STUDENT">
        <li>
            <xsl:value-of select="@LAST" />,
            <xsl:value-of select="@FIRST" />
        </li>
    </xsl:template>

</xsl:stylesheet>
```

- The result of the stylesheet is an HTML page which contains the student names, e.g. "Smith, Ann" as an unordered list.

  `value-of` adds the value of the XPath-expression converted to a string.

Example XSLT Stylesheet (9)

```
<STUDENT SID='101'
         FIRST='Ann' LAST='Smith'
         EMAIL='smith@acm.org'/>
```

```
<xsl:template match="STUDENT">
    <li><xsl:value-of select="@LAST" />,
        <xsl:value-of select="@FIRST" />
    </li>
</xsl:template>
```

```
<li>Smith,
    Ann
</li>
```

Stylesheets are XML (1)

- Note that XSLT stylesheets must be well-formed XML. Thus, even if HTML is generated, one must e.g. write “
” and not simply “
”.
- XML has only the five predefined entities “<”, “>”, “'”, “&quote;”, “&”.
- To use other HTML entities (e.g. “ ”):
  - ◇ declare them in a local DTD part in the DOCTYPE declaration (see below), or
  - ◇ put them into a CDATA section (see below), or
  - ◇ write a character reference: for .

Stylesheets are XML (2)

- Solution with CDATA-Section:

```
<xsl:text disable-output-escaping="yes"
        ><![CDATA[ ]]></xsl:text>
```

- The xsl:text is needed so that the CDATA-section does not appear in the output (because the output escaping is disabled).

  xsl:text simply creates a text node.
  [https://www.w3.org/TR/xslt#section-Creating-Text]
  [https://www.w3.org/TR/xslt#disable-output-escaping]

- Obviously, this is only practical if there are very few entity references.

Stylesheets are XML (3)

```
<!DOCTYPE xsl:stylesheet [
    <!ENTITY Auml  "&#x00C4;">
    <!ENTITY auml  "&#x00E4;">
    <!ENTITY Ouml  "&#x00D6;">
    <!ENTITY ouml  "&#x00F6;">
    <!ENTITY Uuml  "&#x00DC;">
    <!ENTITY uuml  "&#x00FC;">
    <!ENTITY szlig "&#x00DF;">
    <!ENTITY nbsp  " ">
]>
```

The numbers can be taken from the HTML DTD or the Unicode standard or [http://www.w3.org/2003/entities/2007/w3centities-f.ent].

Overview

Templates as Functions (1)

- One can view a template rule as a function with the following input:
  - ◇ a current node,

    The current node is the context node for the evaluation of XPath expressions within the template. It is the main input of the template rule. [http://www.w3.org/TR/xslt#rules]
  - ◇ a current node list,

    The current node list is only used for determining the context position and context size: The current node is always a member of the current node list. Its position is the context position (counted from 1). The length of the current node list is the context size.
  - ◇ possibly named parameters (see below).

Templates as Functions (2)

- The output is a "result tree fragment".

  This is a root node with children. The root node is not important, it is normally removed when the "result tree fragment" is inserted into another "result tree fragment". I.e. it could also be simply formalized as a list of nodes. However, because a root node cannot have attribute nodes as children, a result tree fragment cannot contain attribute nodes (except within element nodes). The special type "result tree fragment" as opposed to node set is also needed because one is not allowed to use /, // and [...] on result tree fragments (only the string value can be used). It is possible to define variables that have result tree fragments as values, (see Slide 8-72), but one cannot do much with it except inserting it into the output document. In particular, one cannot apply templates to the result of other templates.
  [http://www.w3.org/TR/xslt#section-Result-Tree-Fragments]

Templates as Functions (3)

- One can view the template rules as different versions of a heavily overloaded function/method.

  All functions have the same name, they differ only in the argument type. Later, "modes" will be discussed, which correspond to functions with different names. One can also introduce named templates.
- The attribute `match` defines the "data type" of the current node (the argument) for which this "implementation" of `xs:template` is applicable.
- E.g. there may be one implementation for `STUDENT` element nodes, and another one for `EXERCISE` nodes.

Templates as Functions (4)

- Initially, `xs:template` is called with the root node of the input document as current node.

  I.e. this is the call of the "main" procedure which initiates the stylesheet execution. The "current node list" contains only this node, i.e. the context position is 1 and context size is 1.
- When a template is executed, the contents of the `xs:template` element is evaluated.

  This process is described in detail starting on Slide 8-37. The language can be seen as a functional language, i.e. the contents of the `xs:template` is basically a term which is evaluated to a result tree fragment based on the values of its subterms. There are variables (see Slide 8-72), but these can be assigned a value only once, i.e. they are constants.

Templates as Functions (5)

- If `<xsl:apply-templates select="e">` is called,
  - the XPath-expression e is evaluated. The result must be a node set. Let $n_1, \ldots, n_k$ be these nodes in document order (`xs:sort` changes the order).
  - For $i := 1, \ldots, k$, a recursive call of the template evaluation procedure is done with n_i as current node and $n_1, \ldots, n_k$ as current node list.

    I.e. context node: n_i, context position: i, context size: k.
  - Each call returns a sequence of nodes (result tree fragment). These sequences are concatenated (with adjacent text nodes merged).

Templates as Functions (6)

- Within a template, one can e.g. use

  ```
  <xsl:value-of select="position()"/>
  ```

  to see the context position.

  For instance, this can be used to generate a sequential number of the template calls resulting from a single `xsl:apply-templates`. In the same way, `last()` gives the total number of nodes selected in this call.
- But note that the context changes within an XPath expression (and thus, the value of `position()`).

  E.g. “`//STUDENT[position()+1]`” gives $\emptyset$, because `position()` is then the position of the current node in the `//STUDENT`-sequence. If one needs to remember the position for which the template was called, this can be done by defining a variable within the template, see Slide 8-75.

Templates as Functions (7)

- Since the current node (the main input parameter to the template invocation) is so important, there is a special function `current()` which returns it.

  There are several “additional functions” that can be used in XPath expressions embedded in XSLT, which are not part of the XPath core function library. [http://.../xslt#add-func] [#function-current]
- When an XPath expression in the template is evaluated, the context node ist first the current node.
- However, as the path expression navigates through the XML document, the context node changes, while the result of `current()` remains stable.

Templates as Functions (8)

- Note that a template invocation for a node n can call templates for any node in the input document, not only in the subtree rooted at n.
- Print sum of homework points for each student:

```
<xsl:template match="STUDENT">
    <li>
        <xsl:value-of select="@LAST" />:
        <xsl:value-of select="sum(//RESULT
            [@SID=current()/@SID][@CAT='H']
            /@POINTS)" />
    </li>
</xsl:template>
```

Template Instantiation (1)

General Remarks:

- The contents of the `xs:template`-Element is "instantiated" to give a result tree fragment (which usually becomes part of the output document).

  When the "real" XSLT evaluation starts, there is an XDM tree for the input document, and one for the stylesheet (which is XML, too).

- I.e. to specify what XSLT does, one must define a function "`instantiate`" that takes a context C and a node n inside the template as input, and returns a sequence of nodes for the result tree.

  Usually, it will call itself recursively for the children of node n.

Template Instantiation (2)

Literal Result Elements:

- Elements within the template that do not belong to the XSLT namespace are evaluated by creating the corresponding element in the output.

  [http://www.w3.org/TR/xslt#literal-result-element]

- The content of the element in the template is evaluated to give the content of the constructed element.

  Therefore, e.g. nested `xsl:value-of` or `xsl:apply-templates` are evaluated, too.

Template Instantiation (3)

Literal Result Elements, continued:

- Attributes of the element are treated as "attribute value templates": They can contain XPath-expressions in "`{...}`" that are evaluated to give the attribute value of the constructed element.

  This is similar to XQuery. If one needs literal "{" outside an XPath expression, one has to write "{{", and the same for "}". Note that this special interpretation of "`{...}`" happens only in attribute values of literal result elements or certain attributes of some XSLT elements. Within element content, "`{...}`" is not interpreted.
  [http://www.w3.org/TR/xslt#dt-attribute-value-template]

Template Instantiation (4)

Literal Text:

- Text nodes within the template are copied to the generated output, unless they contain only white-space.

  And unless an acestor element of the text node contains the special attribute `xml:space="preserve"` (introduced in the XML standard).
  [http://www.w3.org/TR/xslt#section-Creating-Text]
  [http://www.w3.org/TR/xslt#strip]

- Note that adjacent text nodes (no matter how they are created) are merged, and empty text nodes are removed, as required by the data model.

Template Instantiation (5)

`xsl:text`:

- If one wants to generate e.g. a single space in the output document, one can write

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

  Whitespace within this element is preserved.

- The element `xsl:text` can be used for generating any text node, but because literal text is copied, this is needed only in special situations.
- The content mode of `xsl:text` is `#PCDATA`.

  I.e. pure text without nested elements.

Template Instantiation (6)

`disable-output-escaping`:

- If one writes `<` or `<`, the internal representation of the style sheet contains the character "<".
- Normally, on output this is escaped, i.e. written "`<`", so that valid XML is produced.

  The same is done for "&" and maybe other special characters.

- The elements `xsl:text` and `xsl:value-of` have an attribute `disable-output-escaping` which permits to supress this translation.

  [http://www.w3.org/TR/xslt#disable-output-escaping]

Template Instantiation (7)

`disable-output-escaping`, continued:

- The default value of this attribute is `"no"`.
- If `disable-output-escaping` is set to "`yes`", characters like < are printed without the translation to `<`.
- Note that in the stylesheet, one must write `<` (so that the stylesheet is valid XML).
- But the output will then be simply "<".
- E.g., if one wants to generate "`ä`":

```
<xsl:text disable-output-escaping="yes"
    >&amp;auml;</xsl:text>
```


Template Instantiation (8)

`value-of`:

- `<xsl:value-of select="`e`"/>` is replaced by the value of the XPath-expression e, converted to string.

  The node, for which the template is applied, is the context node for evaluating e. The result of `value-of` is always a string (text node). [https://www.w3.org/TR/xslt#value-of]

- If several nodes are selected, only the first (in document order) is chosen, and its string-value is taken.

  As explained above for `xsl:text`, the attribute `disable-output-escaping` can be used to suppress the translation of characters like < to `<`.

Template Instantiation (9)

`apply-templates`:

- `<xsl:apply-templates select="e"/>` is replaced by the result of doing the "template rule" transformation recursively for all nodes that in the result of the XPath-expression e. See Slide 8-33.
- If `select` is missing, all child nodes are selected.

  `xsl:apply-templates` can contain `xsl:sort` →8-80 and `xsl:with-param` →8-83. Furthermore, it has an attribute `mode` →8-82. [http://www.w3.org/TR/xslt#section-Applying-Template-Rules]
- Of course, a template can contain any number of calls to `xsl:apply-templates`, not only one.

Template Instantiation (10)

`xsl:copy`:

- `<xsl:copy>C</xsl:copy>` copies the current node (the node for which the template was called) and replaces its children by the result of evaluating C.

  I.e. the node is copied, but not its children or attributes. The copied node can be any kind, it does not have to be an element node. [http://www.w3.org/TR/xslt#copying]
- The identity transformation can be specified as:

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


Template Instantiation (11)

`xsl:copy-of`:

- `<xsl:copy-of select="e"/>` copies the result of evaluating e into the result of the template, including all descendant nodes.

  [http://www.w3.org/TR/xslt#copy-of]
- I.e. `xsl:copy-of` can be used to copy portions of the input XDM tree into the output.

  The input XDM tree is subject to the removal of pure whitespace text nodes, as defined by `<xsl:strip-space elements="A B C"/>` (whitespace text nodes that are children of the named elements `A`, `B` and `C` will be removed), see [http://www.w3.org/TR/xslt#strip].

Template Instantiation (12)

`xsl:element`:

- `<xsl:element name="n">C</xsl:element>` generates an element node with name n and content that is the result of evaluating C.

  [http://.../xslt#section-Creating-Elements-with-xsl:element]
- One can use `{e}` in the attribute value of `name` to include XPath-expressions that are evaluated and replaced by the result (converted to a string).

  I.e. the attribute value is interpreted as "attribute value template". This possibility to compute the element name is probably the reason for using `xsl:element`, because otherwise one could have written simply the element itself.

Template Instantiation (13)

`xsl:attribute`:

- `<xsl:attribute name="`n`">`C`</xsl:attribute>` generates an attribute node with name n and the result of evaluating C as value.

  One can use `{`e`}` in the attribute value of `name` (see `xsl:element` above). The result of evaluating C must be a text node (e.g., element nodes cannot become an attribute value). [http://www.w3.org/TR/xslt#creating-attributes]

- The generated attribute node is assigned to the enclosing element.

  Attribute nodes must come first in the content, they cannot be added, e.g., after a text node.

Template Instantiation (14)

Summary:

- There are constructor elements for each node type:
  - ◇ `xsl:element`
  - ◇ `xsl:attribute`
  - ◇ `xsl:text`
  - ◇ `xsl:processing-instruction`

    [http://.../xslt#section-Creating-Processing-Instructions]
  - ◇ `xsl:comment`

    [http://www.w3.org/TR/xslt#section-Creating-Comments]

  Obviously, these are similar to the computed constructors in XQuery. Literal elements, attributes and text in the template are copied to the result, this corresponds to the direct element constructors of XQuery.

Template Rule Selection (1)

- A template with XPath-expression p in the attribute "`match`" is applicable to a node n
  - ◇ if there is some ancestor a of n
  - ◇ such that n is an element of the nodes selected by p evaluated with context node a.
- For instance, "`/GRADES-DB/STUDENT/`" matches
  - ◇ a `STUDENT`-node within the top `GRADES-DB` node,
  - ◇ not a `GRADES-DB` node with a `STUDENT` child node.
- There are priority rules if several templates match.

  See below and [https://www.w3.org/TR/xslt#conflict].

Template Rule Selection (2)

- Only a subset of XPath 1.0 is allowed as pattern, basically a |-union of path expressions using
  - ◇ /, //,

    At the beginning for absolute paths and as separators of the steps.
  - ◇ `child::`, `attribute::`

    Of course, the child axis does not have to be specified explicitly, and the attribute axis can be abbreviated to "`@`".
  - ◇ node tests (name and type tests) and
  - ◇ predicates `[...]`.

    Inside predicates, the full XPath 1.0 can be used. [http://www.w3.org/TR/xslt#patterns]

Template Rule Selection (3)

- Each template has a priority.
- One can specify a priority explicitly:

```
<xsl:template match="STUDENT[1]" priority="2.0">
```

- If one does not specify a priority, the default is:
  - ◇ −0.5 if it is just a node test without a name, e.g. `*` or `@*` or `text()`.
  - ◇ −0.25 if it is a wildcard with a namespace, e.g. `abc:*`
  - ◇ 0 if it is a name, e.g. `STUDENT` or `@EMAIL`.
  - ◇ +0.5 for all more complex patterns.

Template Rule Selection (4)

- If a pattern is composed with |, then the template is treated like several templates, one for each alternative.
- E.g. if the pattern is “`STUDENT|EXERCISE[1]`”, the template rule has priority
  - ◇ 0, if applied to a `STUDENT`-element, and
  - ◇ 0.5, if applied to the first `EXERCISE`-child of its parent.

Template Rule Selection (5)

- A stylesheet can be composed out of several files:

```
<xsl:import href="lib.xsl">
```

- Template rules from files that are imported earlier have lower “import precedence” than templates from later imported files.

  Template rules in the main file have the highest “import precedence”. [`xsl:apply-imports`] can be used for applying overridden rules.

- For template rule selection, “import precedence” is considered first, and priority second.

  It is an error if that still leaves more than one template rule. However, the XSLT processor may recover by selecting the later rule.

Built-In Template Rules (1)

- Built-in template rule for descending in the tree if there is no other match:

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

  When `apply-templates` is specified without the attribute `select`, it processes all child nodes of the current node (element nodes, text nodes, comment nodes and PI nodes).

- Built-in rules have the lowest possible import precedence. Thus, if there is another rule matching the node, this other rule is chosen.

Built-In Template Rules (2)

- The following rule prints text nodes and values of attribute nodes when/if they are selected:

```
<xsl:template match="text()|@*">
    <xsl:value-of select="."/>
</xsl:template>
```

Note that the above default rule applies the templates only to child nodes, not the attribute nodes.

- There is also a rule for processing instructions and comment nodes that returns an empty node set:

```
<xsl:template
     match="processing-instruction()|comment()"/>
```


Possible Non-Termination

- A template can contain `xsl:apply-templates` with a `select`-expression that matches not only for child or descendant nodes, but any node in the input document (including itself).

```
<xsl:template match="STUDENT>
    <em>
        <xsl:apply-templates select="."/>
    </em>
</xsl:template>
```

- XSLT is computationally complete.

One can simulate any Turing machine (or other computation model) in it. [http://www.unidex.com/turing/utm.htm]

Overview

Restrictions in XPath 1.0 (1)

- In this course, the main emphasis is on XPath 2.0.
- However, the XSLT-implementation of many browsers supports only XSLT 1.0.
- Therefore, if one wants to write style sheets which are executed in the browser, one should use only XPath 1.0.
- Axis and location paths are basically the same.

Node type tests have been expanded in XPath 2.0 (because of XML Schema).

Restrictions in XPath 1.0 (2)

- XPath 1.0 is not based on sequences as XPath 2.0. The available data types are:
  - ◇ string
  - ◇ number (floating point)

    There is no integer type in XPath 1.0. Note that the type "number" contains an error value (`NaN`: "not a number"), positive and negative infinity, and a positive and a negative zero.
  - ◇ boolean
  - ◇ node set

    Instead of sequences, XPath 1.0 has only node sets. Often the nodes must be ordered, this is done in document order. Some rules in XPath 2.0 become clearer if one remembers that the designers tried to remain compatible with the node sets from XPath 1.0.

Restrictions in XPath 1.0 (3)

- Variables can be defined only outside the path expression (in XSLT, not in XPath). Therefore, there is no `for`, `some`, `every`, `:=`.
- There is also no `if`.
- `except` und `intersect` are not available in XPath 1.0.

  `union` is also not available, but | is (which does $\cup$).
- Since the type system is much more restriced (not based on XML Schema), there are no type tests of type conversions.

  E.g. no `instance of`, `treat as castable`, `cast`.

Operators in XPath 1.0

Prio	Operator	Assoc.
1	`or`	left
2	`and`	left
3	`=`, `!=`	left
4	`<`, `<=`, `>`, `>=`	left
5	`+`, `-`	left
6	`*`, `div`, `mod`	left
7	`-` (unary)	right
8	`|`	left
9	`/`, `//`	left
10	`[ ]`	left

Functions in XPath 1.0 (1)

Node Set Functions:

- `number last()`: Context size.
- `number position()`: Context position.
- `number count(node-set` x`)`: Number of nodes in x.
- `node-set id(object` x`)`:

  Nodes with one of the IDs specified by x.

  `object` means that x can be of any type. If x is a string containing a single ID, a node with that ID is taken. Otherwise x is converted to a sequence of strings, which can also contain several IDs separated by whitespace.

Functions in XPath 1.0 (2)

Node Set Functions, continued:

- `string local-name(node-set?` x`)`:
  Node name without namespace prefix.
  The first node in the input node set is taken (in document order). The argument is optional. If the function is called without argument, the local name of the context node is returned.
- `string namespace-uri(node-set?` x`)`:
  Namespace URI of (first) argument node.
- `string name(node-set?` x`)`:
  Name of (first) node in x with namespace prefix.

Functions in XPath 1.0 (3)

String Functions:

- `string string(object?` x`)`: Conversion to string.
  For a node set, the string-value of the first node in the set is taken. If the argument is omitted, the context node is taken.
- `string concat(string` s_1`, string` s_2`, string`* s_n`)`:
  String concatenation.
- `boolean starts-with(string` x`, string` y`)`:
  y is prefix of x.
- `boolean contains(string` x`, string` y`)`:
  y is substring of x.

Functions in XPath 1.0 (4)

String Functions, continued:

- `string substring-before(string` x`, string` y`)`:
  Prefix of x until first occurrence of y.
  The empty string is returned if x does not contain y.
  E.g. `substring-before("abcbc", "b") = "a"`.
- `string substring-after(string` x`, string` y`)`:
  Suffix of x after first occurrence of y.
  E.g. `substring-after("abcbc", "b") = "cbc"`.
- `string substring(string` s`, number` p`, number?` l`)`:
  Substring of s starting at position p with length l.
  E.g. `substring("abcde", 2, 3) = "bcd"`. If l is omitted: entire rest.

Functions in XPath 1.0 (5)

String Functions, continued:

- `number string-length(string?` s`)`:
  Number of characters in s.
- `string normalize-space(string?` s`)`:
  s with sequences of whitespace characters reduced to a single space, and leading/trailing whitespace removed.
- `string translate(string` s`, string` x`, string` y`)`:
  s with the i-th character in x replaced by the i-th character in y (or removed if `string-length`$(y) < i$).

Functions in XPath 1.0 (6)

Boolean Functions:

- **boolean boolean(object** x**)**: Conversion to boolean.

  See "effective boolean value": E.g. node set is true iff it is not empty.
- **boolean not(boolean** b**)**: Negation.
- **boolean true()**: Constant value "true".
- **boolean false()**: Constant value "false".
- **boolean lang(string** l**)**:
  Language of the context node is l.

  E.g. if `xml:lang = "en-us"` is specified in an ancestor node of the context node (and no other language is specified in between), `lang("en")` is true. If no language is specified, `lang` returns false.

Functions in XPath 1.0 (7)

Number Functions:

- **number number(object?** x**)**: Conversion to a number.

  If a string has no numeric format, it is converted to the special floating point value "NaN" ("not a number", error value).
- **number sum(node-set** x**)**:
  Sum of the result of converting the string-value of each node in x to a number.
- **number floor(number** x**)**: Largest integer $\leq x$.
- **number ceiling(number** x**)**: Smallest integer $\geq x$.
- **number round(number** x**)**:
  x rounded to nearest integer (.5 is rounded up).

Overview

Variables (1)

- XPath 1.0 has no constructs that can introduce new variables, or assign a value to a variable.
- But one can use `$x` to access the value of a variable `x` defined in the given context.
- In XSLT 1.0, a variable can be declared and assigned a value as follows:

```
<xsl:variable name="version" select="1"/>
```

- One can use this e.g. with

```
<xsl:value-of select="$version"/>
```

Variables (2)

- It is also possible to define the variable value in the content. This is processed as a template:

```
<xsl:variable name="table_headline">
    <tr><th>Student</th><th>Points</th></tr>
</xsl:variable>
```

- One can use this result tree fragment as follows:

```
<xsl:copy-of select="$table_headline"/>
```

- Note: `<xsl:variable name="n" select="2">` is not the same as `<xsl:variable name="n">2</xsl:variable>`.

  In the first case, the value is the number 2, in the second case a node with value 2. E.g. `//student[$n]` works in the first case, but in the second, one must write `//student[position()=$n]`.

Variables (3)

- Variable declarations can appear on the top level, i.e. as child of `xsl:stylesheet`. Then they are global and can be accessed everywhere.

  If they are defined by a template, this is evaluated with the root node of the input document as context node.

- Variable declarations can also be written inside a template, then they are local, and the variable can be accessed only within that template.

  And only after it is defined. This is different from global variables, which are available even before their point of definition (but cycles are forbidden). [http://www.w3.org/TR/xslt#variables]

Variables (4)

- A defined variable cannot be assigned a new value (i.e. it should be understood like a constant).

  A global variable can be shadowed by a local variable in a template.

- This permits very different evaluation algorithms, e.g. on parallel hardware.

  The language is declarative (functional), not imperative.

- Of course, a variable within a template gets a new value for each template invocation:

```
<xsl:template match="STUDENT">
    <xsl:variable name="no" select="position()"/>
```


Repetition: `for-each` (1)

- With the `for-each` construct, one can embed a template directly into another template:

```
<xsl:template match="GRADES-DB">
    <ul>
        <for-each select="STUDENT">
            <li>
                <value-of select="@LAST"/>,
                <value-of select="@FIRST"/>
            </li>
        </for-each>
    </ul>
</xsl:template>
```

  [http://www.w3.org/TR/xslt#for-each]

Repetition: for-each (2)

- The XPath-expression in `select` determines a node set $n_1, \ldots, n_k$, just as with `apply-templates`.
- The contents of the `for-each` element is instantiated once for each node n_i as current node, the corresponding node lists (result tree fragments) are concatenated as the value of the `for-each`.

  Note that there is no variable, only the implicit context.
- If one uses `apply-templates` with a separate template, that becomes a reusable component.

  Like a procedure. In contrast, `for-each` is like putting the procedure body directly into the place where it is called.

Conditional Processing (1)

- `xsl:if` makes part of a template conditional, i.e. the contents of `xsl:if` is inserted only when a condition is true.

  [http://www.w3.org/TR/xslt#section-Conditional-Processing]
- E.g. comma-separated list of student last names:

```
<xsl:template match="GRADES-DB">
    <for-each select="STUDENT">
        <value-of select="@LAST"/>
        <xsl:if test="not(position()=last())"
            >,</xsl:if>
    </for-each>
</xsl:template>
```


Conditional Processing (2)

- An “else if”-chain, e.g.

  “`if(`c_1`) then` s_1 `else if(`c_2`) then` s_2 `else` s_3”

  is written in XSLT as

```
<xsl:choose>
    <xsl:when test="c1">s1</xsl:when>
    <xsl:when test="c2">s2</xsl:when>
    <xsl:otherwise>s3</xsl:otherwise>
</xsl:choose>
```

  The content model of `xsl:choose` is `xsl:when+,xsl:otherwise?`. The template within the first `xsl:when` with a `test`-condition that evaluates to true is chosen (evaluated). If all are false, the content of `xsl:otherwise` is evaluated (if missing, it is treated as empty). [http://.../xslt#section-Conditional-Processing-with-xsl:choose]

Sorting (1)

- One can specify that `apply-tempates` and `for-each` should construct the result not in document order of the selected nodes, but in a specific sort order.
- E.g. print students alphabetically sorted by last name, and by first name if last names are equal:

```
<xsl:template match="GRADES-DB">
    <ul>
        <apply-templates select="STUDENT">
            <sort select="@LAST"/>
            <sort select="@FIRST"/>
        </apply-templates>
    </ul>
</xsl:template>
```

Sorting (2)

- Attributes of `xsl:sort` to modify the sort order:
  - ◇ `data-type="number"` for numerical order.

    The default is `data-type="text"`, which means that the selected values are converted to strings. With `data-type="number"`, they are converted to numbers. All attributes are "attribute value templates": one can use `{...}`. [http://www.w3.org/TR/xslt#sorting]
  - ◇ `order="descending"` means an inverse sort order from large to small values. Default: `"ascending"`.
  - ◇ `lang="de"` select a language-specific sort order.
  - ◇ `case-order="upper-first"` requests a sort order like `A a B b ....` Alternative: `"lower-first"`.

Modes

- `xsl:template` and `xsl:apply-templates` have an optional attribute `mode` which can be set to a name.

  Possibly with namespace prefix (QName). [http://.../xslt#modes]
- `xsl:apply-templates` applies only templates with a matching `mode` value, i.e.
  - ◇ either both have a `mode` value specified, and the values are equal, or
  - ◇ both have no `mode` specified.
- I.e. the mode is something like a function name, and `match` specifies the argument type.

Template Parameters

- One can pass a parameter to a called template:

```
<xsl:apply-templates select="//RESULT[@CAT='E']">
    <xsl:with-param name="type" select="'Exam'"/>
</xsl:apply-templates>
```

  The parameter definition is very similar to a variable definition. [http://.../xslt#section-Passing-Parameters-to-Templates]
- A template that uses the parameter value must declare it at the beginning of the content:

```
<xsl:templates match="RESULT>
    <xsl:param name="type" select="'Unknown'"/>
```

  The defined value is a default value. The parameter can be accessed as a variable `$type` in XPath. [http://www.w3.org/TR/xslt#variables]

Named Templates

- `xsl:template` has an optional attribute `name`. There can be only one template with a given name.

  Templates from imported stylesheets can be overridden. [http://www.w3.org/TR/xslt#section-Defining-Template-Rules]
- One can call a named template as follows:

```
<xsl:call-template name="t">
    <xsl:with-param name="p" select="e"/>
</xsl:call-template>
```

- This does not change the current node.

  In contrast to `xsl:apply-templates`, there is no attribute `select` for specifying a new current node list. So the context is the same as in the calling template. [http://www.w3.org/TR/xslt#named-templates]