XPath

HAP/LAP. Corpus Linguistics.

XPath

- XPath (the XML Path Language): query language for selecting nodes from an XML document
- May be also used to compute values from the content of an XML document
- Defined by the World Wide Web Consortium (W3C): 1.0 (1999), 2.0 (2007, 2010), 3.0 (2014), and 3.1 (2017)
  - version 1.0 is the most widely available today
- Basic component of several XML technologies: XSLT, XLink, XPointer, XQuery, XML Schema...

Outline

What is XPath?

- Not-XML syntax for selecting and referencing XML document nodes (“sections”)
- The document is taken as a tree: XML elements, atrributes, etc. are referenced using “tree *paths*”
- It contains a set of useful functions

XML tree paths

- Similar to directory paths to specify files on a file system

```
C:\Documents\memo.doc
```

```
catalog/cd/title
```

Example: /catalog

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
 <cd country="USA">
   <title>Empire Burlesque</title>
   <artist>Bob Dylan</artist>
   <price>10.90</price>
 </cd>
 <cd country="UK">
   <title>Hide your heart</title>
   <artist>Bonnie Tyler</artist>
   <price>9.90</price>
 </cd>
 <cd country="USA">
   <title>Greatest Hits</title>
   <artist>Dolly Parton</artist>
   <price>9.90</price>
 </cd>
</catalog>
```

Example: /catalog/cd

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd country="USA">
   <title>Empire Burlesque</title>
   <artist>Bob Dylan</artist>
   <price>10.90</price>
  </cd>
  <cd country="UK">
   <title>Hide your heart</title>
   <artist>Bonnie Tyler</artist>
   <price>9.90</price>
  </cd>
  <cd country="USA">
   <title>Greatest Hits</title>
   <artist>Dolly Parton</artist>
   <price>9.90</price>
  </cd>
</catalog>
```

Example: /catalog/cd/price

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

Example: `/catalog`

```
/catalog/cd[price > 10.80]
```

- `cd` elements whose price is above 10.80

Outline

Syntax: node location

- XML document: node tree.
- XPath pattern: a node sequence, separated by slashes
  - `/catalog/cd/price`

    absolute path: `price` elements having `cd` parents and `catalog` grandparents
  - `//cd`

    `cd` elements anywhere in the document

Syntax: using *

- `/catalog/cd/*`

  any element whose parent is `cd` and whose grandparent is `catalog`
- `/catalog/*/price`

  `price` elements, whose grandparent is `catalog`
- `/*/*/price`

  `price` elements having two ancestors
- `//*`

  all document elements

Syntax: filter elements using predicates

- `/catalog/cd[1]`

  first `cd` child of `catalog`
- `/catalog/cd[last()]`

  last `cd` child of `catalog`
- `/catalog/cd[price]`

  `cd` elements whose parent is `catalog` and which contain a `price` child
- `/catalog/cd[price=10.90]`

  `cd` elements whose parent is `catalog` and which contain a `price` child with value 10.90
- `/catalog/cd[price > 10]/title`

  `title` of `cd`'s with `price` above 10

Syntax: many paths in one pattern

- `/catalog/cd/title | /catalog/cd/artist`

  `title` and `artist` elements, whose parents are `cd` and grandparents are `catalog`
- `//title | //artist`

  all `title` and `artist` elements
- `//title | //artist | //price`

  all `title`, `artist` and `price` elements
- `/catalog/cd/title | //artist`

  `title` elements, whose parents are `cd` and grandparents are `catalog`, and all `artist` elements

Syntax: attributes in patterns @

- `//@country`

  all `country` attributes
- `//cd[@country]`

  `cd` elements having a `country` attribute
- `//cd[@*]`

  `cd` elements having at least one attribute
- `//cd[@country='UK']`

  `cd` elements having a `country` attribute with value `UK`

Exercise: which XPath expression?

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

Exercise: which XPath expression?

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

Exercise: which XPath expression?

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

Outline

Location paths

- A path expression selects nodes in a tree (1.0: node-set; 2.0: node sequence)
- Comprises a series of steps (location steps)
- Paths can be:
  - absolute (`/` in the beginning): start at the root element
  - relative (no `/` in the beginning): relative to the previous step
- Location steps are separated by the `/` character:

```
/step/step/...
step/step/...
```

Location paths

- Location steps are evaluated from left to right
- At each step, there is a *context node* against which the step is evaluated
  - the root node (first step in an absolute path)
  - the node-set selected by the previous location step (relative path)

Location paths

- *location step*:
  - *axis*: defines the relationship to be followed in the tree (child nodes, ancestor nodes, etc.)
  - *node test*: defines what nodes are requested

    kind or type test: what kind of nodes (elements, attributes, etc.)
    name test: name of the nodes
  - *predicates* (one or more): provide the ability to filter nodes according to a selection criteria
- Syntax:

  axis::*node-test*[*predicate*]
- Example:

  `child::price[.=0.90]`

Axes and node tests

- The axis specifies a set of nodes relative to the current node (direction)
- The node test selects a set of nodes in this direction, according to its type or its name
- Type node tests:
  - `comment()`: comment nodes
  - `node()`: any node except attributes and namespaces
  - `processing-instruction()`: processing-instructions
  - `text()`: text nodes

Axes

- `ancestor`: ancestors of current node (parents, grandparents, grand grandparents...), also including the root node (if the current node is not the root node itself)
- `ancestor-or-self`: like `ancestor`, but including the current node
- `attribute`: attributes of the current node
- `child`: children of the current node
  - note: attribute nodes are not considered children of element nodes.
- `parent`: parent of current node
  - note: empty if current node is the tree root

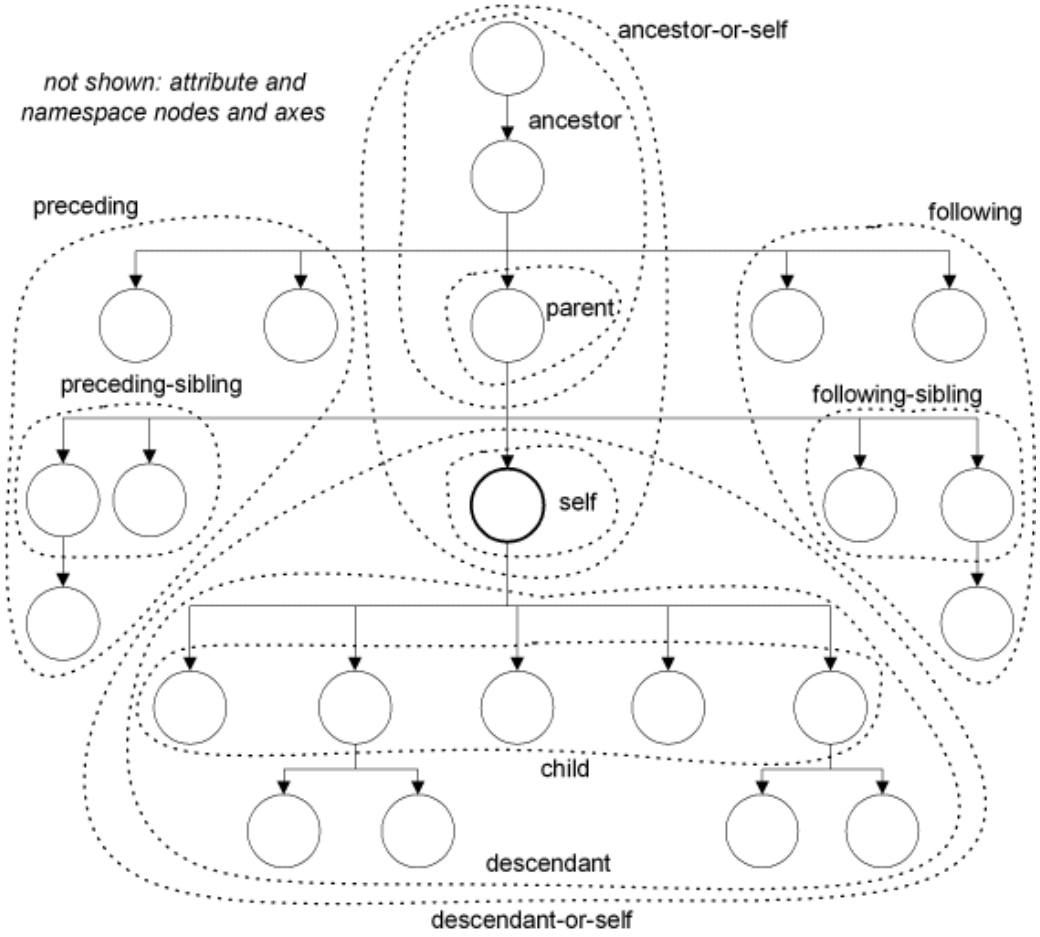
Axes

- `descendant`: descendants of the current node
  - note: it does not consider attribute nodes
- `descendant-or-self`: like `descendant`, but including the current node
- `following`: nodes that appear after the current node in the document, excluding the descendants of current node
- `following-sibling`: nodes that appear after the current node in the document, and that are children of current node's parent

Axes

- `preceding`: nodes that appear before the current node in the document, excluding the ancestors of current node
- `preceding-sibling`: nodes that precede the current node in the document, and that are children of current node's parent
- `self`: the current node itself

Axes, graphically



The union of the `ancestor`, `descendant`, `following`, `preceding` and `self` axes is the whole document. Its intersection is empty.

Order of nodes in axes

- The use of axes considers an XML tree to be ordered (document order)

  the tree is traversed in a pre-order fashion: left to right

- If axis points *forwards* (`following`, `child`...), the order is forward; if axis points *backwards* (`preceding`, `ancestor`...), the order is backward

  to be taken into account when dealing with node positions!

Examples

- `child::cd`

  all `cd` elements that are children of the current node
- `attribute::src`

  `src` attribute of the current element
- `child::*`

  children elements of the current node

Examples

- `attribute::*`
  current element's attributes
- `child::text()`
  text nodes that are children of current node
- `child::node()`
  current node's children
- `descendant::cd`
  `cd` elements that are descendant of the current node

Examples

- `ancestor::cd`

  `cd` elements that are ancestors of the current node
- `ancestor-or-self::cd`

  `cd` elements that are ancestors of the current node, or the current node itself
- `child::*/child::price`

  `price` element that is grandchildren of the current node
- `/`

  document root node

Examples: predicates

- `child::price[text() = 9.90]`

  `price` elements that are children of the current node, and that contain a textual child whose value is 9.90
- `child::cd[position() = 1]`

  first `cd` child of the current node
- `child::cd[position() = last()]`

  last `cd` child of the current node

Examples: predicates

- `child::cd[position() = last() - 1]`

  next to last `cd` child of the current node
- `child::cd[position() < 6]`

  first five `cd` children of the current node
- `child::cd[position() = 7]`

  seventh `cd` child of the current node
- `child::cd[attribute::type = "classic"]`

  `cd` elements that are children of the current node, and that contain a `type` attribute with value “classic”

Examples: abbreviated syntax

- There exists an abbreviated syntax for writing XPath expressions
- For instance, we can discard the `child::` axis specification, as it is the default axis if not specified otherwise

$$\texttt{cd} \equiv \texttt{child::cd}$$

Examples: abbreviated syntax

	`child::`	**`cd`** `child::cd`
@	`attribute::`	**`cd[@type="classic"]`** `child::cd[attribute::type="classic"`

.	`self::node()`	**`.//cd`**
		`self::node()/`
		`descendant-or-self::node()/child::c`

..	`parent::node()`	**`../cd`** `parent::node()/child::cd`

//	`descendant-or-self::node()`	**`//cd`**
		`descendant-or-self::node()/`
		`child::cd`

Examples

- `cd[last()]`
- `*/cd`
- `/book/chapter[3]/para[1]`
- `//cd`
- `.`
- `.//cd`
- `..`
- `../@src`
- `cd[@type="classic"]`
- `cd[@type="classic"][5]`
- `cd[5][@type="classic"]`
- `cd[@type and @country]`

Outline

Expressions

- Arithmetic: `+, -, *, div, mod`
- Equality: `=, !=`
- Relational: `<,<=, >, >=`
- Boolean: `or, and`

Expressions

- Comparison among node-sets
  - `=` is `true` if there exists a node in the node-set for which the condition holds
  - `!=` is `true` if there exists a node in the node-set for which the condition does not hold
- Conclusion: two node-sets can be both equal and different!

Expressions: iteration and variables

- Supported from XPath 2.0 onwards

```
for $n in ./name return concat($n/firstname, '
', $n/surname)
```

  - performs iterations over (node) sequences
  - returns a (node) sequence

  - the variable `$n` above represents the value processed in each iteration and can be used in the `return` part

Functions

- `count(node-set)` → number
- `id(value)` → node-set
- `last()` → number
- `position()` → number
- `name(node)` → string

Supported from XPath 2.0 onwards:

- `distinct-values(sequence)` → sequence (of unique atomic values)

Functions: strings

- `concat(str1, str2...)` → string

  `concat('The', ' ', 'XML')` → `'The XML'`
- `normalize-space(str)` → string

  `normalize-space('The  XML')` → `'The XML'`
- `contains(str1, str2)` → boolean

  `contains('XML', 'X')` → `true`
- `starts-with(str1, str2)` → boolean

  `starts-with('XML', 'X')` → `true`
- `string(value)` → string

  converts any value to a string

  `string(3.09)` → `3.09`
- `string-length(str)` → number

  `string-length('Beatles')` → `7`

Functions: strings

- `substring(str1, begin, [length])` → string
  - `substring('Beatles', 2, 4)` → `eatl`
- `substring-after(str1, str2)` → string
  - `substring-after('12/10', '/')` → `10`
- `substring-before(str1, str2)` → string
  - `substring-before('12/10', '/')` → `12`
- `translate(str1, str2, str3)` → string
  - `translate('12:30', ':', '!')` → `12!30`

Functions: numbers

- `ceiling(number)` → number
  - largest integer value greater than argument.
    `ceiling(3.14)` → `4`
- `floor(number)` → number
  - largest integer value less or equal to argument.
    `floor(3.14)` → `3`
- `round(number)` → number
  - closest integer to the argument
    `round(3.14)` → `3`
- `sum(node-set)` → number
  - sums all numerical values of node-set
    `sum(//price)`