Tutorial 11: XPath
Outline

• XML Document
  – Tree structure

• Xpath Queries
  – Selecting descendants and attributes
  – Functions
  – Shortcuts

• Examples and exam questions
<bib>
  <book price="55">
    <publisher>Addison-Wesley</publisher>
    <author>Serge Abiteboul</author>
    <author>
      <first-name>Rick</first-name>
      <last-name>Hull</last-name>
    </author>
    <author>Victor Vianu</author>
    <title>Foundations of Databases</title>
    <year>1995</year>
  </book>
  <book>
    <!-- this is a comment -->
    <publisher>Freeman</publisher>
    <author>Jeffrey D. Ullman</author>
    <title>Principles of Database and Knowledge Base Systems</title>
    <year>1998</year>
  </book>
</bib>
XML Document as a Tree

Root Node

Root Element

book

price=55

publisher

Addison-Wesley

author

Serge Abiteboul
XPath Query Example

- /bib/book/year
  Result:
  
  
  
  <year>1995</year>
  <year>1998</year>

- /bib/paper/year
  Result:
  Empty
  (No elements named paper)
Descendent or self

• //author

Result:
<author>Serge Abiteboul</author>
<author>
  <first-name>Rick</first-name>
  <last-name>Hull</last-name>
</author>
<author>Victor Vianu</author>
<author>Jeffrey D. Ullman</author>

• /bib//first-name

Result:
<first-name>Rick</first-name>

<bib>
  <book price="55">
    <publisher>Addison-Wesley</publisher>
    <author>Serge Abiteboul</author>
    <author>
      <first-name>Rick</first-name>
      <last-name>Hull</last-name>
    </author>
    <author>Victor Vianu</author>
    <title>Foundations of Databases</title>
    <year>1995</year>
  </book>
  <book>
    <!-- this is a comment -->
    <publisher>Freeman</publisher>
    <author>Jeffrey D. Ullman</author>
    <title>Principles of Database and Knowledge Base Systems</title>
    <year>1998</year>
  </book>
</bib>
Selecting non-element nodes

- `text()`
  - Returns text nodes
- `node()`
  - Returns any node
- `comment()`
  - Returns comment nodes

What `/bib/book/author/text()` will return?

Author names in one text node:
- Serge Abiteboul
- Victor Vianu
- Jeffrey D. Ullman

Rick Hull does not appear because there is no text node under the author element

- `//comment()`
- `<!– this is a comment -->`
Selecting Unknown Nodes

- `//author/*`

Result:

```xml
<first-name>Rick</first-name>
<last-name>Hull</last-name>
```

- The `*` selects every node element
  - Excluding text nodes
Accessing attributes

- /bib/book/@price
- Result: “55”

- @* can be used to select any attribute
Operators

• General comparison operators:
  
  =, !=, >, <, >=, <=

  In case of comparing a sequence with a single value returns true if there is one value matching to required comparison.

• Order comparison:
  
  >>, <<, is

  Comparing the relative order of elements in the document.

  Not Defined for comparing sequences.
Operators

• $a >> b$ returns true
  – if node $a$ appears after node $b$ in the document.

• $<<$ will check the opposite

• For example:
  – //last-name $>>$ //first-name  true
  – //first-name $>>$ //last-name  false
Selecting using Boolean Expressions

• `/bib/book/author[first-name = “Rick”]`

Result:

```xml
<author>
  <first-name>Rick</first-name>
  <last-name>Hull</last-name>
</author>
```
Selecting using Boolean Expressions

• //book[@price<60][publisher = "Addison-Wesley"]
  – the semantic meaning of [..][..] –
    • First filter by the first expression and then
    • filter by the second.

In this case it is equivalent to :
• //book[@price<60 and publisher = "Addison-Wesley"]
  – Not always equivalent!!
  – as we will see later with different Boolean expressions
Existential Boolean Expressions

- `/bib/book/author[first-name][address[zip][city]]/last-name`

Result:
- Last names of authors having
  - first name and
  - address containing zip and city.
- `[first-name]` is a Boolean condition

- Is it possible to return authors having first name only (and not last name)? Yes.
  - `/bib/book/author[first-name][not(last-name)]`
Functions

- The query `/bib/book[2]`
  - Is equivalent to `/bib/book[position() = 2]`

- position() returns the position of the node with respect to its context node.

- `//author[last()]`
  - last() return the position of the last node in the set of nodes with respect to the context node.
Are the following queries equivalent?

- //author[1][2]
- //author[2][1]
Functions - Equivalence(?) Example

- //author[1] returns the first author of each book:

  That is,
  <author>Serge Abiteboul</author>
  <author>Jeffery D. Ullman</author>

- //author[1][2]
  - does not return values
  - Due to the fact that each of the context nodes has only one element

- //author[2] equivalent to //author[2][1]:
  <author>
    <first-name>Rick</first-name>
    <last-name>Hull</last-name>
  </author>
  - In this case, the context node is different and we return the first element.
Functions

• /bib/*[name()="book"]
  – Equivalent to /bib/book
  – name() return the name of the node

• /bib/book[count(author)>1]
  – Return books having more than 1 author.

• id("a0130353000")
  – id() return nodes having the given id as input
<table>
<thead>
<tr>
<th>Axis Name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ancestor</td>
<td>Selects all ancestors (parent, grandparent, etc.) of the current node</td>
</tr>
<tr>
<td>ancestor-or-self</td>
<td>Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself</td>
</tr>
<tr>
<td>attribute</td>
<td>Selects all attributes of the current node</td>
</tr>
<tr>
<td>child</td>
<td>Selects all children of the current node</td>
</tr>
<tr>
<td>descendant</td>
<td>Selects all descendants (children, grandchildren, etc.) of the current node</td>
</tr>
<tr>
<td>descendant-or-self</td>
<td>Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself</td>
</tr>
<tr>
<td>Axes</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>following</td>
<td>Selects everything in the document after the closing tag of the current node</td>
</tr>
<tr>
<td>following-sibling</td>
<td>Selects all siblings after the current node</td>
</tr>
<tr>
<td>parent</td>
<td>Selects the parent of the current node</td>
</tr>
<tr>
<td>preceding</td>
<td>Selects all nodes that appear before the current node in the document, except ancestors, attribute nodes and namespace nodes</td>
</tr>
<tr>
<td>preceding-sibling</td>
<td>Selects all siblings before the current node</td>
</tr>
<tr>
<td>self</td>
<td>Selects the current node</td>
</tr>
<tr>
<td>Shortcut</td>
<td>Shortcut</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>child::</td>
<td>default</td>
</tr>
<tr>
<td>attribute::</td>
<td>@</td>
</tr>
<tr>
<td>self::node()</td>
<td>.</td>
</tr>
<tr>
<td>parent::node()</td>
<td>..</td>
</tr>
<tr>
<td>/descendant-or-self::node()/</td>
<td>//</td>
</tr>
<tr>
<td>[position() = number]</td>
<td>[number]</td>
</tr>
</tbody>
</table>
## Shortcuts - Examples

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>/child::bib/child::book</td>
<td>/bib/book</td>
</tr>
<tr>
<td>/child::bib/child::book/attribute::price</td>
<td>/bib/book/@price</td>
</tr>
<tr>
<td>/self::node()/descendant-or-self::node()/child::title</td>
<td>.//title (equivalent to //title)</td>
</tr>
<tr>
<td>/descendant-or-self::node()/child::author/parent::node()</td>
<td>//author/..</td>
</tr>
</tbody>
</table>
Are the following queries equivalent?

- //author[1]
- //author/.[1]

No! 
//author/.[1] returns all authors in the document because the ./ make every author node to be its own context node.
Examples

//author/descendant-or-self::node()/child::zip
⇔ //author//zip
⇔ //author/descendant::zip

What will be returned in the following?

• //book/publisher/parent::*//author
• //book[author[position()<last()]]
• //book[count(author[1]) = count(author[1] | author[last()]) ]
Examples

• For which books the year of publication is the most recent?
  • //book[ not(year < //book/year) ]
To simplify the display of the DTD, assume that the elements GID, date, name, country, color, no, status, from, to, type are PCDATA.

Write an XPath query that finds the name of the player who moved a piece during move number 7 of game number 99. The query must find the player by comparing the color in which he plays to the color of the tool that was moved during the course of the discussion.
Game 99:

//game[GID=99]

The color of move 7 in game 99:

//game[GID=99]/move[no=7]/piece/color

The name of the player who moved at move 7 in game 99:

//game[GID=99]/player[color = ../move[no=7]/piece/color]/name