In this lecture

- Path Expression
- Regular Expression
- XPath specification
  - data model
  - examples
  - Syntax

Resources:

- Data on the Web Abiteboul, Buneman, Suciu : section 4.1
- A formal semantics of patterns in XSLT by Phil Wadler.
- XML Path Language (XPath) www.w3.org/TR/xpath
Path Expressions

Examples:
- Bib.paper
- Bib.book.publisher
- Bib.paper.author.lastname

Given an OEM instance, the *answer* of a path expression \( p \) is a set of objects.
Path Expressions

Examples:

DB =

\[
\text{Bib.paper} = \{&o12, &o29\}
\]

\[
\text{Bib.book.publisher} = \{&o51\}
\]

\[
\text{Bib.paper.author.lastname} = \{&o71, &206\}
\]
Answer of a Path Expression

Simple evaluation algorithms for Answer(P, DB):

\[
\text{Answer}(P, DB) = f(P, \text{root}(DB))
\]

Where:

\[
f(\varepsilon, x) = \{ x \}
\]

\[
f(L.P, x) = \bigcup \{ f(P, y) \mid \exists (x, L, y) \in \text{edges}(DB) \}
\]

Runs in PTIME in size(P), size(db):
- PTIME complexity
Regular Path Expressions

\[ R ::= \text{label} \mid \_ \mid R.R \mid (R|R) \mid R^* \mid R^+ \mid R? \]

Examples:
- Bib.(paper|book).author
- Bib.book.author.lastname?
- Bib.book.(references)*.author
- Bib.(\_)*.zip
Applications of Regular Path Expressions

Navigating uncertain structure:
- Bib.book.author.lastname?

Syntactic substitute for inheritance:
- Bib.(paper|book).author
- Better: Bib.publication.author, but we don’t have inheritance
Applications of Regular Path Expressions

Computing transitive closure:
- \( \text{Bib.}(_*)\text{.zip} \)
  = everything accessible to zip
- \( \text{Bib.book.}(_*)\text{.author} \)
  = everything accessible via references to author

Some regular expressions of doubtful practical use:
- \( (_*)\text{.}(reference*)\) = a path with an even number of references
- \( (_*)\) = paths of even length
- \( (_*)\text{.}(?)\text{.}(?)\) = paths of length \(3m + 4n\) for some \(m,n\)
Answer of a Regular Path Expression

Recall:
- $\text{Lang}(R) = \text{the set of words } P \text{ generated by } R$

Answer of regular path expressions:
- $\text{Answer}(R, DB) = \bigcup \{ \text{Answer}(P, DB) \mid P \in \text{Lang}(R) \}$

Need an evaluation algorithm that copes with cycles
XPath

- [ ] http://www.w3.org/TR/xpath (11/99)
- [ ] Building block for other W3C standards:
  - XSL Transformations (XSLT)
  - XML Link (XLink)
  - XML Pointer (XPointer)
  - XML Query
- [ ] Was originally part of XSL
Example for XPath Queries

```
<bib>
  <book>  <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 price="55">  
    <publisher> Freeman </publisher>  
    <author> Jeffrey D. Ullman </author>  
    <title> Principles of Database and Knowledge Base Systems </title>  
    <year> 1998 </year>  
  </book>
</bib>
```
Data Model for XPath

Much like the Xquery data model
XPath: Simple Expressions

/bib/book/year

Result:  <year> 1995 </year>
        <year> 1998 </year>

/bib/paper/year

Result: empty (there were no papers)
XPath: Restricted Kleene Closure

//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>
Xpath: Functions

/bib/book/author/text()  

Result:  
Serge Abiteboul  
Victor Vianu  
Jeffrey D. Ullman

Rick Hull doesn’t appear because he has firstname, lastname

Functions in XPath:  
- text() = matches the text value  
- node() = matches any node (= * element or @* attribute or text())  
- name() = returns the name of the current tag
Xpath: Wildcard

//author/*

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

* Matches any element
Xpath: Attribute Nodes

/bib/book/@price

Result: “55”

@price means that price has to be an attribute
Xpath: Qualifiers

/bib/book/author[firstname]

Result: <author> <first-name> Rick </first-name> <last-name> Hull </last-name> </author>
Xpath: More Qualifiers

/bib/book/author[firstname][address[//zip][city]]/lastname

Result: <lastname> … </lastname>
<lastname> … </lastname>
Xpath: More Qualifiers

/bib/book[@price < “60”]

/bib/book[author/@age < “25”]

/bib/book[author/text()]
Xpath: Summary

- `bib` matches a `bib` element
- `*` matches any element
- `/` matches the root element
- `/bib` matches a `bib` element under root
- `bib/paper` matches a `paper` in `bib`
- `bib//paper` matches a `paper` in `bib`, at any depth
- `//paper` matches a `paper` at any depth
- `paper|book` matches a `paper` or a `book`
- `@price` matches a `price` attribute
- `bib/book/@price` matches `price` attribute in `book`, in `bib`
- `bib/book/[@price<"55"]/author/lastname` matches... a last name under `bib/book/author` with book has attribute price <55.
Xpath: More Details

- An Xpath expression, $p$, establishes a relation between:
  - A context node, and
  - A node in the answer set

- In other words, $p$ denotes a function:
  - $S[p] : \text{Nodes} \rightarrow \{\text{Nodes}\}$

- Examples:
  - author/firstname
  - . = self
  - .. = parent
    - part/*/subpart/../.name = part/*/subpart/name
      Any name element in any depth from part that has subpart as parent
The Root and the Root

- `<bib> <paper> 1 </paper> <paper> 2 </paper> </bib>`
- `bib` is the “document element”
- The “root” is above `bib`

- `/bib` = returns the document element
- `/` = returns the root

Why? Because we may have comments before and after `<bib>`; they become siblings of `<bib>`

- This is advanced xmlogy
Xpath: More Details

We can navigate along 13 axes:
  - ancestor
  - ancestor-or-self
  - attribute
  - child
  - descendant
  - descendant-or-self
  - following
  - following-sibling
  - namespace
  - parent
  - preceding
  - preceding-sibling
  - self

We’ve only seen these, so far.
Xpath: More Details

Examples:
- `child::author/child::lastname = author/lastname` : any last name element as child which is a child of author
- `child::author/descendant::zip = author//zip` : any zip elements that is a descendental of author in any depth
- `child::author/parent::* = author/..` :Any parent element which is under (as child of) author
- `child::author/attribute::age = author/@age` : attribute age which is child of author

What does this mean?
- `paper/publisher/parent::*/author`
- `/bib//address[ancestor::book]`
- `/bib//author/ancestor::*//zip = root/bib//author//zip`
Xpath: Even More Details

- `name() = the name of the current node`
  - `/bib//*[name()=book]` same as `/bib//book`

- **What does this mean?**
  - `/bib//*[ancestor::*[name()!=book]]`
    - In a different notation `bib.[^book]*._`
      - Meaning: Any element reachable under `/bib` in any depth except those elements whose ancestor is book element. (Any element reachable under `/bib` in any depth as long as they are not descendant of book elements)

- Navigation axis gives us strictly more power!