Querying XML Data

- Querying XML has two components
  - Selecting data
    - pattern matching on structural & path properties
    - typical selection conditions
  - Construct output, or transform data
    - construct new elements
    - restructure
    - order

Querying XML Data

- XPath = simple navigation through the tree
- XQuery = the SQL of XML
  - next time
- XSLT = recursive traversal
  - will not discuss in class

Querying XML

How do you query a directed graph? a tree?

The standard approach used by many XML, semistructured-data, and object query languages:

- Define some sort of a template describing traversals from the root of the directed graph
- In XML, the basis of this template is called an XPath

XPath is widely used

- XML Schema uses simple XPaths in defining keys and uniqueness constraints
- XQuery
- XSLT
- XLink and XPointer, hyperlinks for XML

XPaths

In its simplest form, an XPath is like a path in a file system:

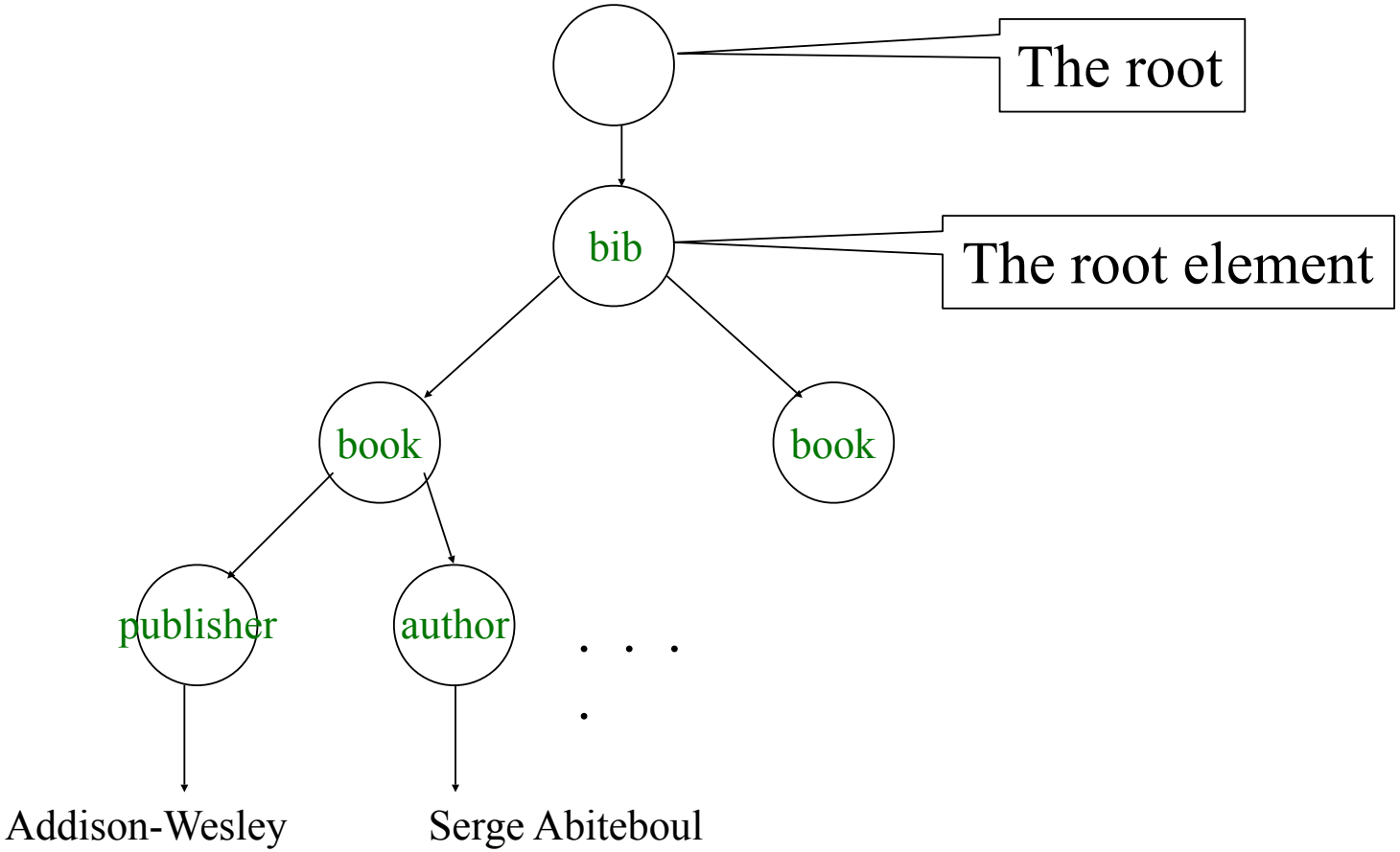
/mypath/subpath/*/morepath

- The XPath returns a **node set** representing the XML nodes (and their subtrees) at the end of the path
- XPaths can have node tests at the end, returning only particular node types, e.g., text(), element(), attribute()
- XPath is fundamentally an ordered language: it can query in order-aware fashion, and it returns nodes in order

Sample Data for 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



XPath

/bib/book/year

/bib/paper/year

//author

/bib//first-name

//author/*

/bib/book/@price

/bib/book/author[firstname]

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

XPath: Simple Expressions

```
/bib/book/year
```

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

```
/bib/paper/year
```

Result: empty (there were no papers)

XPath: descendant axis

```
//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: Text Nodes

```
/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 (= * or @* 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: Predicates

```
/bib/book/author[first-name]
```

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

Xpath: More Predicates

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

Result: <lastname> … </lastname>

<lastname> … </lastname>

Xpath: More Predicates

```
/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…

Axes: More Complex Traversals

Thus far, we've seen XPath expressions that go down the tree

- But we might want to go up, left, right, etc.
- These are expressed with so-called axes:
  - self::path-step
  - child::path-step parent::path-step
  - descendant::path-step ancestor::path-step
  - descendant-or-self::path-step ancestor-or-self::path-step
  - preceding-sibling::path-step following-sibling::path-step
  - preceding::path-step following::path-step
- The previous XPaths we saw were in "abbreviated form"

Context Nodes and Relative Paths

XPath has a notion of a *context* node: it's analogous to a current directory

- "." represents this context node
- ".." represents the parent node
- We can express relative paths:

  subpath/sub-subpath/../.. gets us back to the context node

- By default, the document root is the context node

Predicates – Selection Operations

A *predicate* allows us to filter the node set based on selection-like conditions over sub-XPaths:

/dblp/article[title = “Paper1”]

which is equivalent to:

/dblp/article[./title/text() = “Paper1”]

dot in XPath qualifiers

- //author
- //author[first-name]
- //author[./first-name]
- //author[/first-name]
- //author[//first-name]
- //author[.//first-name]

equivalent

qualifier starts at root

Data Typing in XML

- Data typing in the relational model: schema
- Data typing in XML
  - Much more complex
  - Typing restricts valid trees that can occur
    - theoretical foundation: tree languages
  - Practical methods:
    - DTD (Document Type Descriptor)
    - XML Schema

Document Type Definitions DTD

- Part of the original XML specification
- To be replaced by XML Schema
  - Much more complex
- An XML document may have a DTD
- XML document:

  well-formed = if tags are correctly closed

  Valid = if it has a DTD and conforms to it
- Validation is useful in data exchange

DTD Example

```
<!DOCTYPE company [
  <!ELEMENT company   ((person|product)*)>
  <!ELEMENT person  (ssn, name, office, phone?)>
  <!ELEMENT ssn        (#PCDATA)>
  <!ELEMENT name     (#PCDATA)>
  <!ELEMENT office     (#PCDATA)>
  <!ELEMENT phone    (#PCDATA)>
  <!ELEMENT product  (pid, name, description?)>
  <!ELEMENT pid    (#PCDATA)>
  <!ELEMENT description    (#PCDATA)>
]>
```

DTD Example

Example of **valid** XML document:

```
<company>
   <person>  <ssn> 123456789 </ssn>
             <name> John </name>
             <office> B432 </office>
             <phone> 1234 </phone>
   </person>
   <person>  <ssn> 987654321 </ssn>
             <name> Jim </name>
             <office> B123 </office>
   </person>
   <product> ...  </product>
   ...
</company>
```

DTD: The Content Model

- Content model:
  - Complex = a regular expression over other elements
  - Text-only = #PCDATA
  - Empty = EMPTY
  - Any = ANY
  - Mixed content = (#PCDATA | A | B | C)*

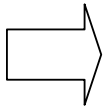
DTD: Regular Expressions

DTD

XML

sequence

```
<!ELEMENT name
                (firstName, lastName))
```



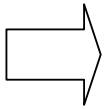
```
<name>
   <firstName>  . . . . . </firstName>
   <lastName>  . . . . . </lastName>
</name>
```

optional

```
<!ELEMENT name  (firstName?, lastName))
```

Kleene star

```
<!ELEMENT person   (name, phone*))
```



```
<person>
   <name>  . . . . . </name>
   <phone>  . . . . . </phone>
   <phone>  . . . . . </phone>
   <phone>  . . . . . </phone>
   . . . . . .
</person>
```

alternation

```
<!ELEMENT person   (name, (phone|email)))
```

Attributes in DTDs

```
<!ELEMENT person  (ssn, name, office, phone?)>
<!ATTLIST    person age CDATA  #REQUIRED>
```

```
<person age="25">
      <name> ....</name>
       ...
</person>
```

Attributes in DTDs

```
<!ELEMENT person  (ssn, name, office, phone?)>
<!ATTLIST    person age          CDATA  #REQUIRED
                    id           ID       #REQUIRED
                    manager    IDREF   #REQUIRED
                    manages    IDREFS   #REQUIRED
>
```

```
<person age="25"
        id="p29432"
        manager="p48293"  manages="p34982 p423234">
    <name> ....</name>
     ...
</person>
```

Attributes in DTDs

Types:

- CDATA = string
- ID = key
- IDREF = foreign key
- IDREFS = foreign keys separated by space
- (Monday I Wednesday I Friday) = enumeration

Attributes in DTDs

Kind:

- #REQUIRED
- #IMPLIED = optional
- value = default value
- value #FIXED = the only value allowed

Using DTDs

- Must include in the XML document
- Either include the entire DTD:
  - <!DOCTYPE rootElement [.....]>
- Or include a reference to it:
  - <!DOCTYPE rootElement SYSTEM “http://www.mydtd.org”>
- Or mix the two... (e.g. to override the external definition)

DTDs Aren't Expressive Enough

DTDs capture grammatical structure, but have some drawbacks:

- Not themselves in XML – inconvenient to build tools for them
- Don't capture database datatypes' domains
- IDs aren't a good implementation of keys
- No way of defining OO-like inheritance

XML Schema

Aims to address the shortcomings of DTDs

- XML syntax
- Can define keys using XPaths
- Subclassing
- Domains and built-in datatypes

Basics of XML Schema

Need to use the XML Schema namespace (generally named xsd)

- simpleTypes are a way of restricting domains on scalars
  - Can define a simpleType based on integer, with values within a particular range
- complexTypes are a way of defining element/attribute structures
  - Basically equivalent to !ELEMENT, but more powerful
  - Specify sequence, choice between child elements
  - Specify minOccurs and maxOccurs (default 1)
- Must associate an element/attribute with a simpleType, or an element with a complexType

Simple Schema Example

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="mastersthesis" type="ThesisType"/>
<xsd:complexType name="ThesisType">
    <xsd:attribute name="mdate" type="xsd:date"/>
    <xsd:attribute name="key" type="xsd:string"/>
    <xsd:attribute name="advisor" type="xsd:string"/>
    <xsd:sequence>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="year" type="xsd:integer"/>
        <xsd:element name="school" type="xsd:string"/>
        <xsd:element name="committeemember" type="CommitteeType"
          minOccurs="0"/>

    </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```