Advanced XML / Data on the Web
Lecture 2

Lars Birkedal [birkedal@it-c.dk]

The IT University of Copenhagen

Outline of this lecture

- Review of XML data model (node-labelled graphs) + DTDs
  - Chapter 3 of ABS
  - XML Revolution Chapter 3
  - Bosak: XML, Java and the Future of the Web
- Comparison of XML with semistructured data
- XSLT
  - XML Revolution Chapter 4–5 (assumed known)
  - Wadler: A formal semantics of patterns in XSLT

XML

- a W3C standard to complement HTML
- descendant of SGML
- motivation:
  - HTML describes layout
  - XML describes content
- HTML 4.0 $\in$ XML $\subseteq$ SGML

HTML

```
<h1>Bibliography</h1>
<p><i>Foundations of Databases</i>
   Abiteboul, Hull, Vianu
   <br>Addison Wesley, 1995
<p><i>Data on the Web</i>
   Abiteboul, Buneman, Suciu
   <br>Morgan Kaufmann, 1999
```

XML

```
<bibliography>
<book>
  <title>Foundations of Databases</title>
  <author>Abiteboul</author>
  <author>Hull</author>
  <author>Vianu</author>
  <publisher>Addison Wesley</publisher>
  <year>1995</year>
</book>
...
```

XML describes the content

XML Terminology

- tags: `book`, `title`, `author`, `...`
- start tag: `<book>`, end tag: `</book>`
- elements: `<book> ... </book>`
  `<author> ... </author>`
- empty element: `<red></red>`, abbr. `<red/>`
- an XML document: a single root element
- *well-formed* XML document: if it has matching tags

More XML Motivation

- Extensibility (not a fixed set of tags)
- Structure (documents can be nested to any level of complexity)
- Validation (XML documents can contain an optional description of its grammar for use by applications that need to perform structural validation — more about this later)

Example XML Application

- LaCoMoCo: Laboratory for Context-dependent Mobile Communication
- Location-based services
  - approaching train, get info about next train towards home if after normal working hours
  - in supermarket need info from Consumer Information
  - distribution of traffic information collected by DR
- we'll use XML for exchange of data

See Bosak article for other examples (health care, aerospace industry, etc.)

XML — Attributes

```
<book price="55" currency="USD">
  <title>Foundations of Databases</title>
  <author>Abiteboul</author>
  ....
  <year>1995</year>
</book>
```

attributes are alternative ways to represent data

XML — Graph Model

```
<person><name>Alice</name>
        <age>42</age>
        <email>a@itu.dk</email>
</person>
```

Node-labelled graphs.

XML vs. SSD I

- ssd expression is a set of label/subtree pairs; denote edge-labelled graphs
- xml element has just one top-level label and then an ordered list of sub-elements; denote node-labelled graphs

XML vs. SSD II

- ssd model is based on unordered collections
- xml elements are ordered:

```
<person><fname>Alice</fname>
        <lname>Smith</lname></person>

<person><lname>Smith</lname>
        <fname>Alice</fname></person>
```

are differenct XML documents

XML vs. SSD III

- *but* XML attributes are not ordered:

  ```
  <person fname="Alice" lname="Smith"/>
  <person lname="Smith" fname="Alice"/>
  ```

  are equal XML documents.
- No order is important for optimizations
- Applications of XML for data exchange likely to ignore order.

XML vs. SSD III

- XML can mix text and elements

```
<talk>XML, Java, and the future of the Web
  <speaker>Jon Bosak</speaker>
</talk>
```

- XML has lots of other stuff (entities, processing instructions, etc.)
- This makes XML data management harder!

XML — Oids and References

- For representation of graphs (not just trees) in XML

```
<person id="o17">
  <name>Alice</name>
</person>
<person id="o78">
  <name>Bob</name>
  <children idref="o12 o13"/>
</person>
<person id="o12">
  <name>Charlie</name>
  <father idref="o78"/>
</person>
```

- oids and references in XML are just syntax

XML — CDATA section

- Syntax: `<![CDATA[`...any text here...`]]>`
- Example:

```
<example>
  <![CDATA[ some text here </notatag><>]]>
</example>
```

XML — Entity References

- Syntax: `&entityname;`
- Example: `<element>less than <</element>`
- Some entities

<	<
>	>
&	&
'	‘
"	“
&	unicode char

XML — Comments

- Syntax: <!--...comment text...-->
- Part of the data model!

XML — Processing Instructions

- Syntax: <? target argument ?>
- Example:

```
<product><name>alarm clock</name>
         <?ringBell 20?>
         <price>19,95</price>
</product>
```

- No semantics, just passed on to the applications processing the data.

XML — Namespaces

- `http://www.w3.org/TR/REC-xml-names/` (January 1999)
- name ::= [prefix] localpart
- prefix is an URI (does not need to actually point to anything), i.e., unique and persistent over time
- Example:

```
<book xmlns:isbn="www.isbn-org.org/def">
  <title>...</title>
  <number>15</number>
  <isbn:number>...</isbn:number>
</book>
```

Used by XSL processors, for example.

DTDs

- Document Type Definition
- Serves as grammar for the underlying XML document
- Part of the XML Language
- Can, to some extent, serve as schema for data represented by XML document (we'll discuss to what extent later)

DTD Example

A DTD for

```
<db><person><name>Alice</name>
            <age>42</age>
            <email>a@itu.dk</email>
    </person>
    <person> ... </person> </db>
```

may look like

```
<!DOCTYPE db [
  <!ELEMENT db (person*)>
  <!ELEMENT person (name,age,email)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT age (#PCDATA)>
  <!ELEMENT email (#PCDATA)>  ]>
```

DTDs as Grammars

- Regular expressions: `e*`, `e+`, `e?`, `e|e'`, `e,e'`
- Context-free grammar for the document, in particular prescribes the *order* of elements
- Can be recursive:

```
<!ELEMENT node (leaf | (node,node))>
<!ELEMENT leaf (#PCDATA)>
```

Declaring Attributes in DTDs

- `<!ATTLIS` element-name attr-name attr-type attr-default`/>`
- Attribute types:
  - `CDATA`: any data allowed (default)
  - `(`value`|...)`: enumeration of allowed values
  - `ID`, `IDREF`, `IDREFS`: `ID` attribute values must be unique, `IDREF` must match some `ID` (reference to an element)

Declaring Attributes in DTDs: Example

```
<!DOCTYPE family [
  <!ELEMENT family (person)*>
  <!ELEMENT person (name)>
  <!ELEMENT name (#PCDATA)>
  <!ATTLIST person id ID #REQUIRED
                   mother IDREF #IMPLIED
                   father IDREF #IMPLIED
                   children IDREFS #IMPLIED>
]>
```

DTDs as Schemas

- not a very rich language for types (compared to programming language and database traditions), e.g., enumerations are not enough for attributes
- complicated to represent non-ordered data
- too simple id attribute mechanism (no point-to requirements (e.g., requirement about the type of the referenced element)
- See XML Revolution Chapter 3 for more comments.
- Later in the course we'll discuss schema languages in more detail.

Break

- 20 minutes break

Formal Semantics of Patterns in XSLT

Recall from Webprogramming (XML Revolution Ch. 5) XSLT

- template rule = pattern + template
- processing by
  - pattern matching
  - instantiating templates
- Now: look at formal semantics of patterns, formulated using standard techniques from programming language semantics.
- Wadler: A formal semantics of patterns in XSLT.

Motivation for Formal Semantics

- Clear and consise description (one page of formulas instead of 10 pages of prose)
- Brings ambiguities to light. E.g., [XSL-Dec98]:

  The result of *MatchExpr* is true if, for any node in the document that contains the context of the *MatchExpr*, the result of evaluating the *SelectExpr* with that node as context contains the context of the *MatchExpr*. Otherwise the result is false

- Should it be “any node in (the document that contains the context of *MatchExpr*)” or “(any node in the document) that contains the context of *MatchExpr*)” ?

Motivation for Formal Semantics

- Makes it possible to *prove* basic properties
- Based on proved properties, can suggest simpler definitions
- E.g., partly based on the semantics presented here, the definition of match patterns of XSLT was simplified and made easier to implement.

Basic definitions

```
<Adam>
  <Cain>
    <Enoch/>
  </Cain>
  <Abel/>
  <Seth>
    <Enosh/>
  </Seth>
</Adam>
```

Let {`Root`, `Adam`, `Cain`, `Enoch`, `Abel`, `Seth`, `Enosh`} be the set of nodes in the tree, where `Root` is a distinguished root node.

Basic definitions

- write $Set(A)$ for the type of a set of elements of type A, i.e., the powerset of the set A
- $children : Node \rightarrow Set(Node)$
- $parent : Node \rightarrow Set_1(Node)$
- Examples:
  - $\texttt{Cain} \in children(\texttt{Adam})$
  - $\{\texttt{Cain}, \texttt{Abel}\} \cup \{\texttt{Seth}\} = children(\texttt{Adam})$

Basic definitions

- $siblings : Node \rightarrow Set(Node)$
- $siblings(x) = \{z \mid y \in parent(x), z \in children(y)\}$
- $siblings(\texttt{Cain}) = \{\texttt{Cain}, \texttt{Abel}, \texttt{Seth}\}$
- $properSiblings : Node \rightarrow Set(Node)$
- $siblings(x) = \{y \mid y \in siblings(x), y \neq x\}$
- $properSiblings(\texttt{Cain}) = \{\texttt{Abel}, \texttt{Seth}\}$
- $grandparent : Node \rightarrow Set_1(Node)$
- $grandparent(x) = \{z \mid y \in parent(x), z \in parent(y)\}$
- $grandparent(\texttt{Adam}) = \emptyset$

Basic definitions

- $\mathit{self}(x) = \{x\}$
- for $r : \mathit{Node} \rightarrow \mathit{Set}(\mathit{Node})$ define
  - $r^+(x) = \{z \mid y \in r(x), z \in r^*(y)\}$
  - $r^*(x) = \{x\} \cup r^+(x)$
- Assume given functions

$$\mathit{first}, \mathit{last} : \mathit{Set}(\mathit{node}) \longrightarrow \mathit{Set}_1(\mathit{Node})$$

- Example: $\mathit{first}(\mathit{children}(\texttt{Adam})) = \{\texttt{Cain}\}$

Data Model for XML

- $isRoot : Node \longrightarrow Boolean$
- $isElement : Node \longrightarrow Boolean$
- $isAttribute : Node \longrightarrow Boolean$
- $isText : Node \longrightarrow Boolean$
- $isComment : Node \longrightarrow Boolean$
- $isPI Node \longrightarrow Boolean$
- $children : Node \longrightarrow Set(Node)$
- $parent : Node \longrightarrow Set_1(Node)$
- $attributes : Node \longrightarrow Set(Node)$
- $Root : Node \longrightarrow Node$

Basic Laws

- $children(x) \neq \emptyset$ implies $isRoot(x) \vee isElement(x)$
- $attributes(x) \neq \emptyset$ implies $isElement(x)$
- $y \in attributes(x)$ implies $isAttribute(y)$
- $y = Root(x))$ implies $isRoot(y)$

Subnodes

- $subnodes : Node \longrightarrow Set(Node)$
- $subnodes(x) = children(x) \cup attributes(x)$
- $y \in subnodes(x)$ iff $parent(y) = x$
- Fact: $x \in subnodes^*(Root(x))$

Name and Value

- $name : Node \longrightarrow String$
- $value : Node \longrightarrow String$

	$name$	$value$
$Root$	empty	value of children
$Element$	tag name	value of children
$Attribute$	attribute name	attribute value
$Text$	empty	content of text node
$Comment$	empty	content of comment
PI	target	content excluding target

Id and Split

Assume given

- $id : String \rightarrow Set_1(Node)$ (returns node associated with unique identifier)
- $split : String \rightarrow Set(String)$ (splits a string at whitespaces)

Denotational Semantics

- done on the board

Abstract Syntax for Patterns

- $n : name$ (also a string)
- $s : String$
- $p ::= p_1 \mid p_2 \mid /p \mid //p \mid \ldots$
- $q ::= q_1$ and $q_2 \mid \ldots$

Semantic Functions

- $M : Pattern \longrightarrow Node \longrightarrow Boolean$
- $S : Pattern \longrightarrow Node \longrightarrow Set(Node)$
- $Q : Qualifier \longrightarrow Node \longrightarrow Boolean$

Lessons from the semantics

- Recall

  The result of *MatchExpr* is true if, for any node in the document that contains the context of the *MatchExpr*, the result of evaluating the *SelectExpr* with that node as context contains the context of the *MatchExpr*. Otherwise the result is false

- See definition of M: pattern p matches in a context x if there is *any* node x_1 in the document such that selecting with pattern p in context x_1 yields the original node x

Lessons from the semantics

Homework Question:

- Explain (in more detail than is given in the paper) the point of the second paragraph of Section 6, page 10, of Wadler's paper A Formal Semantics for Patterns in XSLT. Why would the alternative definition not work ?

Propositions

- done on the board