Processing of structured documents

Spring 2003, Part 1
Helena Ahonen-Myka

Course organization

- 581290-5 laudatur course, 2 cu
- lectures (in Finnish)
  - 21.1.-20.2. Tue 12-14, Thu 10-12, A217
  - not obligatory
- exercise sessions
  - 27.1.-28.2. Mon 16-18, Tue 14-16, C454
  - course assistant: Olli Lahti
  - not obligatory
  - project work included

Requirements

- Exam (Thu 6.3. at 16-20): 45 points
- Project (deadline Fri 14.3.): 15 points
  - integrated into the exercise sessions
  - obligatory to return a report; attending the exercise sessions voluntary
- Maximum of points: 60

Outline (preliminary)

- 1. Structure representations
  - grammatical descriptions
  - data model issues, information sets
  - (XML DTD,) XML Schema
- 2. Processing, transferring XML data
  - SAX, DOM
  - Web services (SOAP, WSDL, UDDI)

Outline...

- 3. Traversing and querying structured documents
  - XPath
  - XML Query
- 4. XML Linking
- 5. Metadata: RDF

Prerequisites

- You should know the basics of XML
  - DTD, elements, attributes, syntax
  - XSLT (basics), formatting
- some programming experience is needed

Project work

- Project work is integrated into the weekly exercises
- A "large" example that lets us play with the concepts and tools discussed in the course
- Each exercise session includes one subtask
  - solution is discussed in the exercise session
- Solutions to the subtasks have to be presented as a report (written in HTML)
- Return a report by 14.3. (as a URL; instructions are given later)

7

1. Structure descriptions

- Regular expressions, context-free grammars -> What is XML?
- (XML Document type definitions)
- data modelling, information sets
- XML Schema

8

Regular expressions

- A way to describe a set of strings over an alphabet (of chars, events, elements…)
- many uses:
  - text searching (e.g. emacs, grep, perl)
  - in grammatical formalisms (e.g. XML DTDs)
- relevant for document structures: what kind of structural content is allowed for different document components

9

Regular expressions

- A regular expression over alphabet Σ is either
  - $\emptyset$ (an empty set)
  - ε (epsilon; sometimes lambda λ)
  - a, where $a \in \Sigma$
  - R | S (choice; sometimes $R \cup S$)
  - R S (catenation) or
  - R* (Kleene closure)
- where R and S are regular expressions

10

Regular expressions

- Regular expression E denotes a language (a set of strings) L(E):
  - $L(\emptyset) = \emptyset$ (empty set)
  - $L(\varepsilon) = \{\varepsilon\}$ (singleton set of empty string)
  - $L(a) = \{a\}$ (singleton set of $a \in \Sigma$)
  - $L(R|S) = L(R) \cup L(S) = \{w \mid w \in L(R) \text{ or } w \in L(S)\}$
  - $L(RS) = L(R)L(S) = \{xy \mid x \in L(R) \text{ and } y \in L(S)\}$
  - $L(R^*) = L(R)^* = \{x_1 \ldots x_n \mid x_k \in L(R), k=1,\ldots,n; n \geq 0\}$

11

Example

- structure of an article:
  - Σ = {title, author, date, section}
  - title followed by an optional list of authors, followed by an optional date, followed by one or more sections:
  - title author* (date | ε) section section*
- common abbreviations:
  - E? = (E | ε); E^+ = E E*
  - -> title author* date? section^+

12

L(title author* date? section+) includes:

title author date section section section

title section

title author author section

Expressive power of regular expressions

- operations:
  - Catenation -> sequential order
  - Choice -> also optional parts
  - Closure -> repetition, optional repetition
- Operations can be nested -> more complex expressions
- … but we cannot express nested structures -> context-free grammars

14

```
<collection>
<article>
    <title></title>
    <author></author><author></author>
    <date></date>
    <sect></sect><sect></sect>
</article>
<article>
    <title></title><section></section>
</article>
</collection>
```

Context-free grammars

- Used widely for syntax specification (programming languages)
- G = (V, Σ, P, S)
  - V: the alphabet of the grammar G; V = $\Sigma \cup N$
  - Σ : the set of terminal symbols;
    N = V- Σ: the set of nonterminal symbols
  - P: set of productions
  - S $\in$ N: the start symbol

16

Productions and derivations

- Productions: A -> α, where A $\in$ N, $\alpha \in$ V*
  - e.g. A -> aBa (1)
- Let γ, $\delta \in$ V*. String γ **derives** δ **directly**, γ => δ, if
  - $\gamma = \alpha A \beta$, $\delta = \alpha\omega\beta$ for some $\alpha,\beta \in$ V*, and A -> ω is a production of the grammar
  - e.g. AA => AaBa (assuming prod. 1 above)

17

Language generated by a context-free grammar

- γ **derives** δ, γ =>* δ, if there is a sequence of 0 or more direct derivations that transforms γ to δ
- The **language generated by** a CFG G:
  - L(G) = {w $\in \Sigma^*$ | S =>* w}
- L(G) is a set of strings: to model structural elements, we consider parse trees

18

Parse trees of a CFG

- Aka syntax trees or derivation trees
- nodes labelled by symbols of V (or by ε):
  - internal nodes by nonterminals, root by start symbol
  - leaves using terminal symbols (or ε)
- parent with label A can have children labeled by $X_1,\ldots,X_k$ only if A -> $X_1\ldots X_k$ is a production

19

CFGs for document structures

- Nonterminals represent document structures
  - e.g. Ref -> AuthorList Title PublData
    AuthorList -> Author AuthorList
    AuthorList -> ε
- problem:
  - obscures the relation of elements (the last Author several hierarchical levels away from Ref) -> solution: extended CFGs

20

Extended CFGs (ECFGs)

- Like CFGs, but right-hand-sides of productions are regular expressions over V, e.g. Ref -> Author* Title PublData
- Let $\gamma, \delta \in V^*$. String γ derives δ directly, $\gamma \Rightarrow \delta$, if
  - $\gamma = \alpha A \beta$, $\delta = \alpha\omega\beta$ for some $\alpha,\beta \in V^*$, and A -> E is a production such that $\omega \in L(E)$
  - e.g. Ref => Author Author Author Title PublData

21

Language generated by an ECFG

- Defined similarly to CFGs
- Theorem: Languages generated by extended and ordinary CFGs are the same -> expressive power is the same

22

Parse trees of an ECFG

- Similar to parse trees of an ordinary CFG, except that..
- parent with label A can have children labeled by $X_1,\ldots,X_k$ when A -> E is a production such that $X_1\ldots X_k \in L(E)$
- -> an internal node may have arbitrarily many children (e.g. Authors below a Ref node)

23

What is XML?

- metalanguage that can be used to define markup languages
  - gives syntax for defining extended context free grammars (DTDs)
  - XML documents that adhere to an ECFG are strings in that language
  - document types (grammars)- document instances (strings in the language)

24

XML encoding of structure

- XML document is essentially a parenthesized linear encoding of a parse tree
  - corresponds to a preorder walk
  - start of inner node (element) A denoted by a start tag <A>, end denoted by end tag </A>
  - leaves are content strings (or empty elements)
- + certain extensions (especially attributes)
- + certain restrictions

25

Terminal symbols in practice

- Leaves of parse trees are normally labeled by single characters (symbols of Σ)
- too granular in practice for XML documents: instead, terminal symbols which stand for all values of a type
  - e.g. #PCDATA in XML for variable length content of data characters
  - richer data types in other XML schema formalisms

26

An example DTD

```
<!DOCTYPE invoice  [
<!ELEMENT invoice    (orderDate, shipDate, billingAddress
                      voice*, fax?)>
<!ELEMENT orderDate (#PCDATA)>
<!ELEMENT shipDate   (#PCDATA)>
<!ELEMENT billingAddress  (name, street, city, state, zip)>
<!ELEMENT voice   (#PCDATA)>
<!ELEMENT fax     (#PCDATA)>
<!ELEMENT name    (#PCDATA)>
<!ELEMENT street  (#PCDATA)>
<!ELEMENT city    (#PCDATA)>
<!ELEMENT state   (#PCDATA)>
<!ELEMENT zip     (#PCDATA)>]>
```

27

And a document:

```
<invoice>
  <orderDate>19990121</orderDate>
  <shipDate>19990125</shipDate>
  <billingAddress>
     <name>Ashok Malhotra</name>
     <street>123 IBM Ave.</street>
     <city>Hawthorne</city>
     <state>NY</state>
     <zip>10532-0000</zip>
  </billingAddress>
  <voice>555-1234</voice>
  <fax>555-4321</fax>
</invoice>
```

28

Context-free vs. context-sensitive

- DTDs describe context-free languages
  - e.g. element orderDate has always the same structure
- Some other schema declaration languages allow context-sensitive structures
  - e.g. orderDate could be different for different products
  - or text paragraph could have different structure restrictions in normal text and in a footnote

29