Context Free Grammars

Context Free Languages (CFL)

- The pumping lemma showed there are languages that are not regular
  - There are many classes "larger" than that of regular languages
  - One of these classes are called "Context Free" languages
- Described by Context-Free Grammars (CFG)
  - Why named context-free?
  - Property that we can substitute strings for variables regardless of context (implies context sensitive languages exist)
- CFG's are useful in many applications
  - Describing syntax of programming languages
  - Parsing
  - Structure of documents, e.g.XML
- Analogy of the day:
  - DFA:Regular Expression as Pushdown Automata : CFG

CFG Example

- Language of palindromes
  - We can easily show using the pumping lemma that the language $L = \{ w \mid w = w^R \}$ is not regular.
  - However, we can describe this language by the following context-free grammar over the alphabet {0,1}:

$P \rightarrow \varepsilon$
$P \rightarrow 0$
$P \rightarrow 1$
$P \rightarrow 0P0$ Inductive definition
$P \rightarrow 1P1$

More compactly: $P \rightarrow \varepsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$

Formal Definition of a CFG

- There is a finite set of symbols that form the strings, i.e. there is a finite alphabet. The alphabet symbols are called **terminals** (think of a parse tree)
- There is a finite set of **variables**, sometimes called non-terminals or syntactic categories. Each variable represents a language (i.e. a set of strings).
  - In the palindrome example, the only variable is P.
- One of the variables is the **start symbol**. Other variables may exist to help define the language.
- There is a finite set of **productions** or production rules that represent the recursive definition of the language. Each production is defined:
  1. Has a single variable that is being defined to the left of the production
  2. Has the production symbol →
  3. Has a string of zero or more terminals or variables, called the body of the production. To form strings we can substitute each variable's production in for the body where it appears.

CFG Notation

- A CFG G may then be represented by these four components, denoted G=(V,T,R,S)
  - V is the set of variables
  - T is the set of terminals
  - R is the set of production rules
  - S is the start symbol.

Sample CFG

1. E→I // Expression is an identifier
2. E→E+E // Add two expressions
3. E→E*E // Multiply two expressions
4. E→(E) // Add parenthesis
5. I→ L // Identifier is a Letter
6. I→ ID // Identifier + Digit
7. I→ IL // Identifier + Letter
8. D → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 // Digits
9. L → a | b | c | … A | B | … Z // Letters

Note Identifiers are regular; could describe as (letter)(letter + digit)*

Recursive Inference

- The process of coming up with strings that satisfy individual productions and then concatenating them together according to more general rules is called *recursive inference*.
- This is a bottom-up process
- For example, parsing the identifier "r5"
  - Rule 8 tells us that D → 5
  - Rule 9 tells us that L → r
  - Rule 5 tells us that I→L so I→r
  - Apply recursive inference using rule 6 for I→ID and get
    - I → rD.
    - Use D→5 to get I→r5.
  - Finally, we know from rule 1 that E→I, so r5 is also an expression.

Recursive Inference Exercise

- Show the recursive inference for arriving at *(x+y1)*y* is an expression

1. E→I
2. E→E+E
3. E→E*E
4. E→(E)
5. I→ L
6. I→ ID
7. I→ IL
8. D → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
9. L → a | b | c | … A | B | … Z

Derivation

- Similar to recursive inference, but top-down instead of bottom-up
  - Expand start symbol first and work way down in such a way that it matches the input string
- For example, given a*(a+b1) we can derive this by:
  - $E \Rightarrow E*E \Rightarrow I*E \Rightarrow L*E \Rightarrow a*E \Rightarrow a*(E) \Rightarrow a*(E+E) \Rightarrow a*(I+E) \Rightarrow a*(L+E) \Rightarrow a*(a+E) \Rightarrow a*(a+I) \Rightarrow a*(a+ID) \Rightarrow a*(a+LD) \Rightarrow a*(a+bD) \Rightarrow a*(a+b1)$
- Note that at each step of the productions we could have chosen any one of the variables to replace with a more specific rule.

Multiple Derivation

- We saw an example of $\Rightarrow$ in deriving a*(a+b1)
- We could have used $\Rightarrow$* to condense the derivation.
  - E.g. we could just go straight to $E \Rightarrow^* E*(E+E)$ or even straight to the final step
    - $E \Rightarrow^* a*(a+b1)$
    - Going straight to the end is not recommended on a homework or exam problem if you are supposed to show the derivation

Leftmost Derivation

- In the previous example we used a derivation called a *leftmost derivation*. We can specifically denote a leftmost derivation using the subscript "lm", as in:

  $\Rightarrow_{lm}$ or $\Rightarrow^*_{lm}$

- A leftmost derivation is simply one in which we replace the leftmost variable in a production body by one of its production bodies first, and then work our way from left to right.

Rightmost Derivation

- Not surprisingly, we also have a rightmost derivation which we can specifically denote via:
- $\Rightarrow_{rm}$ or $\Rightarrow^*_{rm}$
- A rightmost derivation is one in which we replace the rightmost variable by one of its production bodies first, and then work our way from right to left.

Rightmost Derivation Example

- a*(a+b1) was already shown previously using a leftmost derivation.
- We can also come up with a rightmost derivation, but we must make replacements in different order:
  - $E \Rightarrow_{rm} E*E \Rightarrow_{rm} E * (E) \Rightarrow_{rm} E*(E+E) \Rightarrow_{rm} E*(E+I)$ $\Rightarrow_{rm} E*(E+ID) \Rightarrow_{rm} E*(E+I1) \Rightarrow_{rm} E*(E+L1) \Rightarrow_{rm}$ $E*(E+b1) \Rightarrow_{rm} E*(I+b1) \Rightarrow_{rm} E*(L+b1) \Rightarrow_{rm} E*(a+b1)$ $\Rightarrow_{rm} I*(a+b1) \Rightarrow_{rm} L*(a+b1) \Rightarrow_{rm} a*(a+b1)$

Left or Right?

- Does it matter which method you use?
- Answer: No
- Any derivation has an equivalent leftmost and rightmost derivation. That is, $A \Rightarrow^* \alpha$. iff $A \Rightarrow^*_{lm} \alpha$ and $A \Rightarrow^*_{rm} \alpha$.

Language of a Context Free Grammar

- The language that is represented by a CFG G(V,T,P,S) may be denoted by L(G), is a Context Free Language (CFL) and consists of terminal strings that have derivations from the start symbol:

  $L(G) = \{ w \text{ in } T \mid S \Rightarrow^*_G w \}$

- Note that the CFL L(G) consists solely of terminals from G.

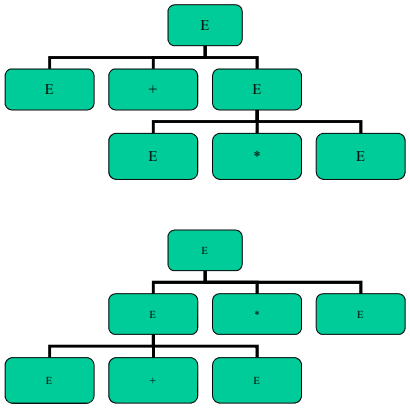
CFG Exercises

Give a CFG for the CFL: $\{0^n 1^n \mid n \geq 1 \}$

Give a CFG for the CFL: $\{a^i b^j c^k \mid i \neq j \text{ or } j \neq k \}$

Ambiguous Grammars

- A CFG is ambiguous if one or more terminal strings have multiple leftmost derivations from the start symbol.
  - Equivalently: multiple rightmost derivations, or multiple parse trees.
- Examples
  - E→ E+E | E*E
  - E+E*E can be parsed as
    - E⇒E+E ⇒E+E*E
    - E ⇒E*E ⇒E+E*E



Ambiguous Grammar

- Is the following grammar ambiguous?
  - S→AS | ε
  - A→A1 | 0A1 | 01

Removing Ambiguity

- No algorithm can tell us if an **arbitrary** CFG is ambiguous in the first place
  - Halting / Post Correspondence Problem
- Why care?
  - Ambiguity can be a problem in things like programming languages where we want agreement between the programmer and compiler over what happens
- Solutions
  - Apply precedence
  - e.g. Instead of: E→ E+E | E*E
  - Use: E→ T | E + T, T→ F | T * F
    - This rule says we apply + rule before the * rule (which means we multiply first before adding)

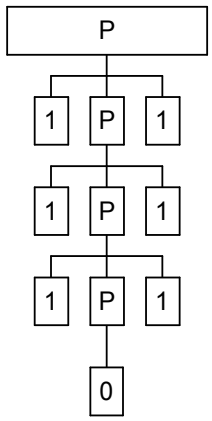
Parse Trees

- A parse tree is a top-down representation of a derivation
  - Good way to visualize the derivation process
  - Will also be useful for some proofs coming up!
- If we can generate multiple parse trees then that means that there is ambiguity in the language
  - This is often undesirable, for example, in a programming language we would not like the computer to interpret a line of code in a way different than what the programmer intends.
  - But sometimes an unambiguous language is difficult or impossible to avoid.

Sample Parse Tree

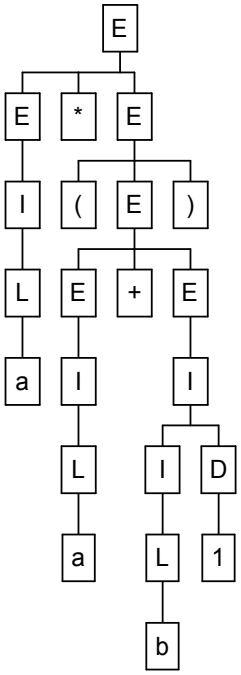
- Sample parse tree for the palindrome CFG for 1110111:

  $P \rightarrow \varepsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$



Sample Parse Tree

- Using a leftmost derivation generates the parse tree for a*(a+b1)
- Does using a rightmost derivation produce a different tree?
- The **yield** of the parse tree is the string that results when we concatenate the leaves from left to right (e.g., doing a leftmost depth first search).
  - The yield is always a string that is derived from the root and is guaranteed to be a string in the language L.



Applications of Context Free Grammars

Introduction to XML

Example 1: Parsing Programming Languages

- Consider an arbitrary expression
  - Arbitrary nesting of operators
  - Parenthesis balancing
  - Requires CFG
- YACC – Yet Another Compiler Compiler
  - Unix program often used to generate a parser for a compiler
  - Output is code that implements an automaton capable of parsing the defined grammar
  - Also mechanisms to perform error handling, recovery

YACC

- Definitions
  - Variables, types, terminals, non-terminals
- Grammar Productions
  - Production rules
  - Semantic actions corresponding to rules
- Typically used with lex
  - Lexical rules → lex → C program with yylex()
    - yylex processes tokens
  - Grammar rules, yylex → yacc → C program with yyparse()
    - yyparse processes grammar of tokens

YACC Example Productions

```
Exp     : ID                    {…}
        | Exp '+' Exp           {…}
        | Exp '*' Exp           {…}
        | '(' Exp ')'           {…}
;
Id      : 'a'                   {…}
        | 'b'                   {…}
        | Id 'a'                {…}
        | Id 'b'                {…}
        | Id '0'                {…}
        | Id '1'                {…}
;
```

{…} contains semantic actions.

Grammar matches:
E→ID | E+E | E*E | (E)
ID→a | b | ID a | ID b |
ID 0 | ID 1

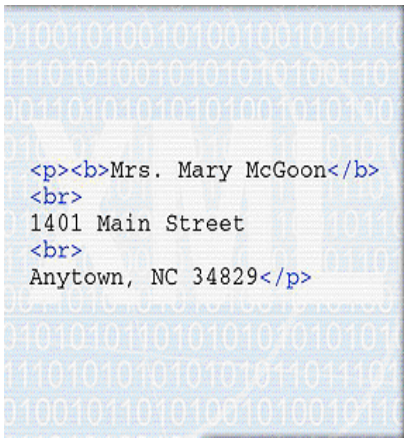
Example YACC Semantics

- | Exp ‘+’ Exp {$$ = $1 + $2}
- | Exp ‘*’ Exp {$$ = $1 * $2}

Example 2: XML - What is it?

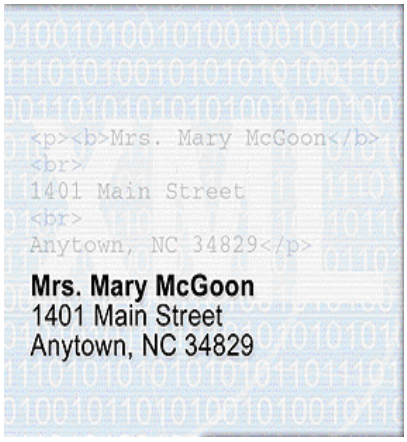
- XML = eXtensible Markup Language
- Technology for web applications - 1997
- World Wide Web Consortium (W3C) standard that lets you create your own tags.
  - Implications for business-to-business transactions on the web.

HTML and XML



- Why do we need XML? We have HTML today
- All browsers read HTML
- Designed for reading by Humans
- Example on the left

HTML Rendered



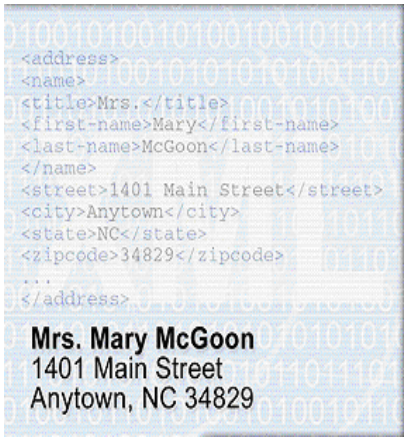
- HTML "rendered" as shown to the left
- Tags describe how the HTML should be displayed, or presented
- Tags don't describe what anything is!

Sample XML File

```
<address>
<name>
<title>Mrs.</title>
<first-name>Mary</first-name>
<last-name>McGoon</last-name>
</name>
<street>1401 Main Street</street>
<city>Anytown</city>
<state>NC</state>
<zipcode>34829</zipcode>
...
</address>
```

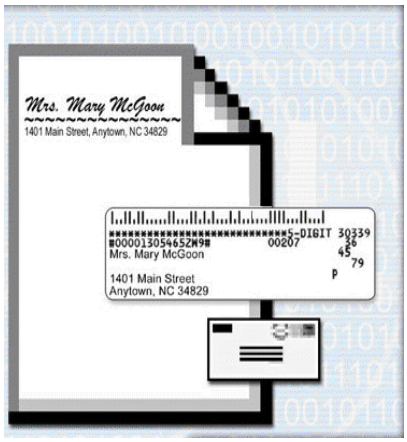
- Same data, but in an XML format
- Humans, but particularly computers, can understand the meaning of the tags
- If we want to know the last name, we know exactly where to look!

Displaying XML



- XML can be rendered, or displayed, just like the HTML page if we so desire
- Rendering instructions aren't stored in the same file, but in a separate XSL file - exTensible Stylesheet Language

Second Rendering



- With a different style sheet, we can render the data in an entirely different way
- Same content, just different presentation

Second example: Song Lyrics in HTML

```
<H1>Hot Cop</H1>
<i> by Jacques Morali, Henri Belolo, and Victor Willis</i>
<ul>
<li>Producer: Jacques Morali
<li>Publisher: PolyGram Records
<li>Length: 6:20
<li>Written: 1978
<li>Artist: Village People
</ul>
```

Song Lyrics in XML

```
<SONG>
  <TITLE>Hot Cop</TITLE>
  <COMPOSER>Jacques Morali</COMPOSER>
  <COMPOSER>Henri Belolo</COMPOSER>
  <COMPOSER>Victor Willis</COMPOSER>
  <PRODUCER>Jacques Morali</PRODUCER>
  <PUBLISHER>PolyGram Records</PUBLISHER>
  <LENGTH>6:20</LENGTH>
  <YEAR>1978</YEAR>
  <ARTIST>Village People</ARTIST>
</SONG>
```

Song XSL Style Sheet for Formatting

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <html> <head><title>Song</title></head>
      <body><xsl:value-of select="."/></body>
    </html>
  </xsl:template>
<xsl:template match="TITLE">
    <h1><xsl:value-of select="."/></h1>
</xsl:template>
</xsl:stylesheet>
```

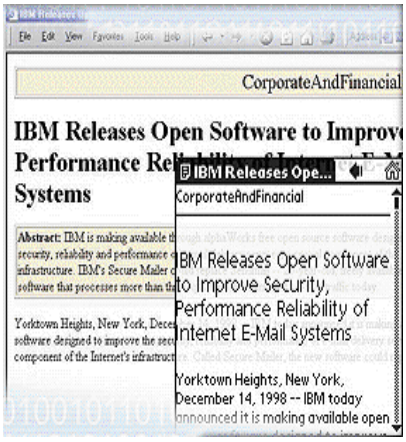
Style Sheets can be quite complex; most translate to HTML

Third Example - News Story

- News article in XML format using the "News" DTD (Document Type Definition – the definition for the allowable tags)

```
<?xmlversion="1.0"?>
<!DOCTYPE NewsArticle SYSTEM "News.dtd">
<NewsArticle>
<Head>
<Title>IBM Releases Open Software to Improve
Security, Performance, and Reliability of
Internet E-Mail Systems</Title>
<Date>12/14/98</Date>
<Summary>IBM is making available through
alphaWorks free open source software
designed to improve the security,
reliability and performance of
e-mail delivery services.</Summary>
<Category topic="Corporate And Financial"/>
...
</NewsArticle>
```

Different Display using Different Style Sheets for Different Apps

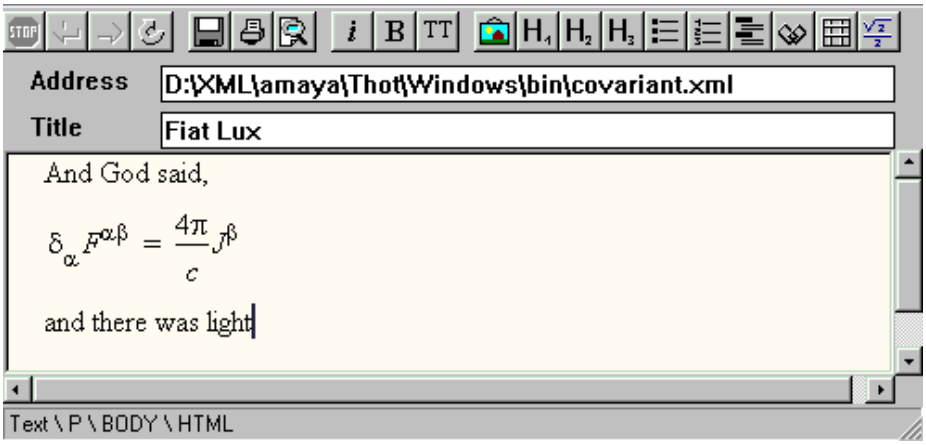


- Desktop rendering using IE
- Palmtop rendering
- Different output needed using different devices, but the same underlying content

Example Applications

- Web Pages
  - XHTML is XML with an HTML DTD
- Mathematical Equations
- Music Notation
- Vector Graphics
- Metadata

Mathematical Markup Language



Vector Graphics

- Vector Markup Language (VML)
  - Internet Explorer 5.0+
  - Microsoft Office 2000+
- Scalable Vector Graphics (SVG)

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">

<rect x="20" y="20" rx="20" ry="20" width="250"
height="100"
style="fill:red;stroke:black;stroke-width:5;opacity:0.5"/>

</svg>
```

File Formats, In-House, Other

- Microsoft Office
- Many Web API's
- RSS uses XML

- Core technology all over the net

Summary of XML Benefits

- Can now send structured data across the web
  - Semantics and Syntax (Presentation), separated
- Business to Business Transactions
  - Using a published XML format (DTD), we can specify orders, items, requests, and pretty much anything we want and display them using any XSL
  - Intelligent Agents can now understand what data means, instead of complex algorithms and heuristics to guess what the data means
    - e.g. Shopping Agents
- Smart Searches using XML queries, not keywords

Where do the XML Tags Come From?

- You get to invent the tags!
- Tags get defined in the DTD (Document Type Definition)

- HTML has fixed tags and presentation meaning only
- XML has user-defined tags and semantic meaning separated from presentation meaning

HTML is a fixed standard. XML lets everyone define the data structures they need.

DTD - Defining Tags

- A **D**ocument **T**ype **D**efinition describes the elements and attributes that may appear in a document
- a list of the elements, tags, attributes, and entities contained in a document, and their relationship to each other - consider it to be a template
- XML documents must be **validated** to ensure they conform to the DTD specs
  - Ensures that data is correct before feeding it into a program
  - Ensure that a format is followed
  - Establish what must be supported
  - E.g., HTML allows non-matching <p> tags, but this would be an error in XML

Sample DTD and XML

greeting.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="greeting.xsl"?>
<!DOCTYPE GREETING SYSTEM "greeting.dtd">
<GREETING>
Hello World!
</GREETING>
```

greeting.dtd

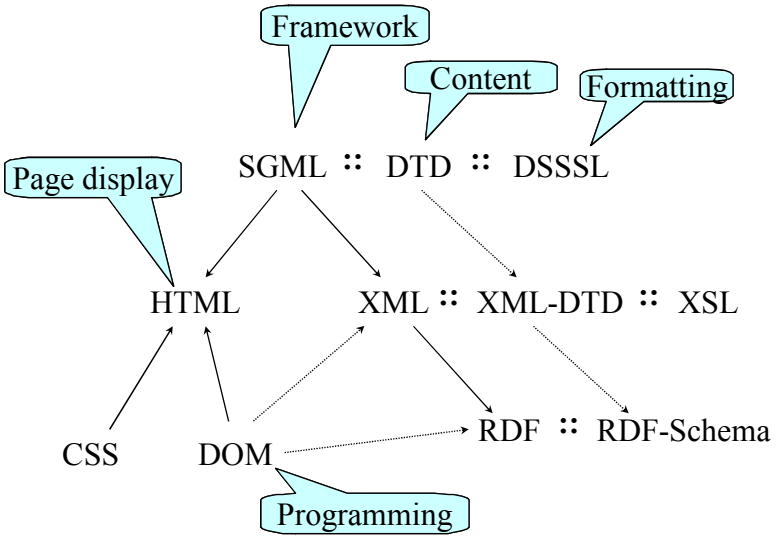
```
<!ELEMENT GREETING (#PCDATA)>
```

Greeting XSL

greeting.xsl

```
<?xml version="1.0"?>
<!--XSLT 1.0 -->
<xsl:transform
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version="1.0">
<xsl:output method="xml" omit-xml-declaration="yes"/>
<xsl:template match="/">
  <H2><xsl:value-of select="greeting"/></H2>
</xsl:template>
</xsl:transform>
```

Family Tree - Derived from SGML (Standard Gen. Markup Lang)



XML Usage

Text Encoding Initiative (TEI)
Channel Definition Format, CDF (Based on XML)
W3C Document Object Model (DOM), Level 1 Specification
Web Collections using XML
Meta Content Framework Using XML (MCF)
XML-Data
Namespaces in XML
Resource Description Framework (RDF)
The Australia New Zealand Land Information Council (ANZLIC) - Metadata
Alexandria Digital Library Project
XML Metadata Interchange Format (XMI) - Object Management Group (OMG)
Educom Instructional Management Systems Project
Structured Graph Format (SGF)
Legal XML Working Group
Web Standards Project (WSP)
HTML Threading - Use of HTML in Email
XLF (Extensible Log Format) Initiative
WAP Wireless Markup Language Specification
HTTP Distribution and Replication Protocol (DRP)
Chemical Markup Language
Bioinformatic Sequence Markup Language (BSML)
BIOpolymer Markup Language (BIOML)
Virtual Hyperglossary (VHG)
Weather Observation Definition Format (OMF)
Open Financial Exchange (OFX/OFE)
Open Trading Protocol (OTP)
Signed XML (W3C)
Digital Receipt Infrastructure Initiative
Digest Values for DOM (DOMHASH)
Signed Document Markup Language (SDML)
FIXML - A Markup Language for the FIX Application Message Layer
Bank Internet Payment System (BIPS)
OpenMLS - Real Estate DTD Design
Customer Support Consortium
XML for the Automotive Industry - SAE J2008
X-ACT - XML Active Content Technologies Council
Mathematical Markup Language
OpenTag Markup
Metadata - PICS
CDIF XML-Based Transfer Format
Synchronized Multimedia Integration Language (SMIL)
Precision Graphics Markup Language (PGML)
Vector Markup Language (VML)
WebBroker: Distributed Object Communication on the Web
Web Interface Definition Language (WIDL)
XML/EDI - Electronic Data Interchange
XML/EDI Repository Working Group
European XML/EDI Pilot Project
EEMA EDI/EC Work Group - XML/EDI
DISA, ANSI ASC X12/XML
Information and Content Exchange (ICE)
CommerceNet Industry Initiative
eCo Framework Project and Working Group
vCard Electronic Business Card
iCalendar XML DTD

More XML Usage

Telecommunications Interchange Markup (TIM, TCIF/IPI)
Encoded Archival Description (EAD)
UML eXchange Format (UXF)
Translation Memory eXchange (TMX)
Scripting News in XML
Coins: Tightly Coupled JavaBeans and XML Elements
DMTF Common Information Model (CIM)
Process Interchange Format XML (PIF-XML)
Ontology and Conceptual Knowledge Markup Languages
Astronomical Markup Language
Astronomical Instrument Markup Language (AIML)
GedML: [GEDCOM] Genealogical Data in XML
Newspaper Association of America (NAA) - Standard for Classified Advertising Data
News Industry Text Format (NITF)
Java Help API
Cold Fusion Markup Language (CFML)
Document Content Description for XML (DCD)
XSchema
Document Definition Markup Language (DDML)
WEBDAV (IETF 'Extensions for Distributed Authoring and Versioning on the World Wide Web')
Tutorial Markup Language (TML)
Development Markup Language (DML)
VXML Forum (Voice Extensible Markup Language Forum)
VoxML Markup Language
SABLE: A Standard for Text-to-Speech Synthesis Markup
Java Speech Markup Language (JSML)
SpeechML
XML and VRML (Virtual Reality Modeling Language)
XML for Workflow Management [NIST]
SWAP - Simple Workflow Access Protocol
Theological Markup Language (ThML)
XML-F ('XML for FAX')
Extensible Forms Description Language (XFDL)
Broadcast Hypertext Markup Language (BHTML)
IEEE LTSC XML Ad Hoc Group
Open Settlement Protocol (OSP) - ETSI/TIPHON
WDDX - Web Distributed Data Exchange
Common Business Library (CBL)
Open Applications Group - OAGIS
Schema for Object-oriented XML (SOX)
XMLTP.Org - XML Transfer Protocol
The XML Bookmark Exchange Language (XBEL)
Simple Object Definition Language (SODL) and XMOP Service
XML-HR Initiative - Human Resources
ECMData - Electronic Component Manufacturer Data Sheet Inventory Specification
Bean Markup Language (BML)
Chinese XML Now!
MOS-X (Media Object Server - XML)
FLBC (Formal Language for Business Communication) and KQML
ISO 12083 XML DTDs
Extensible User Interface Language (XUL)
Commerce XML (cXML)
Process Specification Language (PSL) and XML
XML DTD for Phone Books
Using XML for RFCs
Schools Interoperability Framework (SIF)

XML Query Language

- Several proposals for query language
  - XQuery 1.0 latest?
- Modeling after existing OODB QLs
  - inline construction of XML from XML

```
WHERE <book>
        <publisher><name>Addison-Wesley</></>
        <title> $t</>
        <author> $a</>
      </> IN "www.a.b.c/bib.xml"
CONSTRUCT <result>
            <author> $a</>
            <title> $t</>
          </>
```

  - APIs for script usage

Programming XML

- XML defines an object/attribute data model
- DOM (Document Object Model) is the API for programs to act upon object/attribute data models
  - DHTML is DOM for HTML
    - interface for operating on the document as paragraphs, images, links, etc
    - Programmed with JavaScript, VBScript, modern IDEs often construct much of this for you
  - DOM-XML is DOM for XML
    - interface for operating on the "document" as objects and parameters

Style Sheets / DTD / XML

- The actual XML, Style Sheets, and the DTD (Document Type Definition) could be made by hand, but more typically are created with the help of XML Tools
  - Many tools on the market
  - IBM alphaworks
  - Visual Studio
  - oXygen

Lots of people using it…but

- Not useful if major parties don't agree on a XML format; without adoption everyone has their own format
- Downside: Web full of gobbledygook that only a select few understand
- Even though your browser may parse XML, it may not understand what it really means
- Effect: Everyone can invent their own language on the web
  - Tower of Babel on the web, or Balkanization

Quick Quiz

- What's a DTD?
- Difference between XML and HTML?
- What's a eXtended Style Sheet?
- How can XML make searching easier?

Summary

- XML specifies semantics, not just presentation
  - Semantics separate from Presentation language
  - Users can define their own tags/languages
- Greatly simplifies machine understanding of data
  - Agents easier to implement
  - Business to business transactions
- International, standard format to share and exchange knowledge

Back to Context-Free Grammars…

- HTML can be described by classes of text
  - *Text* is any string of characters literally interpreted (i.e. there are no tags, user-text)
  - *Char* is any single character legal in HTML tags
  - *Element* is
    - Text or
    - A pair of matching tags and the document between them, or
    - Unmatched tag followed by a document
  - *Doc* is sequences of elements
  - *ListItem* is the <LI> tag followed by a document followed by </LI>
  - *List* is a sequence of zero or more list items

HTML Grammar

- Char → a | A | …
- Text → ε | Char Text
- Doc → ε | Element Doc
- Element → Text | <EM> Doc </EM> | <P> Doc | <OL> List </OL>
- ListItem → <LI> Doc </LI>
- List → ε | ListItem List

XML's DTD

- The DTD lets us define our own grammar
- Context-free grammar notation, also using regular expressions
- Form of DTD:

  <!DOCTYPE name-of-DTD [

  list of element definitions

  ]>
- Element definition:
  - <!ELEMENT element-name (description of element)>

Element Description

- Element descriptions are regular expressions
- Basis
  - Other element names
  - #PCDATA, standing for any TEXT
- Operators
  - | for union
  - , for concatenation
  - * for Star
  - ? for zero or one occurrence of
  - + for one or more occurrences of

PC Specs DTD

```
<!DOCTYPE PcSpecs [
        <!ELEMENT PCS (PC*)>
        <!ELEMENT PC (MODEL, PRICE, PROC, RAM, DISK+)>
        <!ELEMENT MODEL (#PCDATA)>
        <!ELEMENT PRICE  (#PCDATA)>
        <!ELEMENT PROC (MANF, MODEL, SPEED)>
        <!ELEMENT MANF (#PCDATA)>
        <!ELEMENT SPEED (#PCDATA)>
        <!ELEMENT RAM (#PCDATA)>
        <!ELEMENT DISK (HARDDISK | CD | DVD )>
        <!ELEMENT HARDDISK (MANF, MODEL, SIZE)>
        <!ELEMENT SIZE (#PCDATA)>
        <!ELEMENT CD (SPEED)>
        <!ELEMENT DVD (SPEED)>
]>
```

Pc Specs XML Document

```
<PCS>
<PC>
        <MODEL>4560</MODEL>
        <PRICE>$2295</PRICE>
        <PROCESSOR>
                <MANF>Intel</MANF>
                <MODEL>Pentium</MODEL>
                <SPEED>4Ghz</SPEED>
        </PROCESSOR>
        <RAM>8192</RAM>
        <DISK>
                <HARDDISK>
                        <MANF>Maxtor</MANF>
                        <MODEL>Diamond</MODEL>
                        <SIZE>2000Gb</SIZE>
                </HARDDISK>
        </DISK>
        <DISK><CD><SPEED>32x</SPEED></CD></DISK>
</PC>
<PC> ….. </PC>
</PCS>
```

Examples with Style Sheet

- Hello world with Greeting DTD
- Product / Inventory List

Prod.XML

```
<?xml version="1.0"?><!--prod.xml-->
<?xml-stylesheet type="text/xsl" href="prodlst.xsl"?>
<!DOCTYPE sales [
<!ELEMENT sales ( products, record )>          <!--sales information-->
<!ELEMENT products ( product+ )>              <!--product record-->
<!ELEMENT product ( #PCDATA )>                <!--product information-->
<!ATTLIST product id ID #REQUIRED>
<!ELEMENT record ( cust+ )>                   <!--sales record-->
<!ELEMENT cust ( prodsale+ )>                 <!--customer sales record-->
<!ATTLIST cust num CDATA #REQUIRED>           <!--customer number-->
<!ELEMENT prodsale ( #PCDATA )>               <!--product sale record-->
<!ATTLIST prodsale idref IDREF #REQUIRED>
]>
<sales>
  <products><product id="p1">Packing Boxes</product>
          <product id="p2">Packing Tape</product></products>
  <record><cust num="C1001">
          <prodsale idref="p1">100</prodsale>
          <prodsale idref="p2">200</prodsale></cust>
        <cust num="C1002">
          <prodsale idref="p2">50</prodsale></cust>
        <cust num="C1003">
          <prodsale idref="p1">75</prodsale>
          <prodsale idref="p2">15</prodsale></cust></record>
</sales>
```

ProdLst.XSL

```
<?xml version="1.0"?><!--prodlst.xsl-->
<!--XSLT 1.0  -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              version="1.0">

<xsl:template match="/">                      <!--root rule-->
  <html><head><title>Record of Sales</title></head>
    <body><h2>Record of Sales</h2>
      <xsl:apply-templates select="/sales/record"/>
    </body></html></xsl:template>
<xsl:template match="record">                 <!--processing for each record-->
  <ul><xsl:apply-templates/></ul></xsl:template>
<xsl:template match="prodsale">               <!--processing for each sale-->
  <li><xsl:value-of select="../@num"/>        <!--use parent's attr-->
      <xsl:text> - </xsl:text>
      <xsl:value-of select="id(@idref)"/>     <!--go indirect-->
      <xsl:text> - </xsl:text>
      <xsl:value-of select="."/></li></xsl:template>
</xsl:stylesheet>
```

ProdTbl.xsl

```
<?xml version="1.0"?><!--prodtbl.xsl-->
<!--XSLT 1.0 -->
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xsl:version="1.0">
 <head><title>Product Sales Summary</title></head>
 <body><h2>Product Sales Summary</h2>
  <table summary="Product Sales Summary" border="1">
                                                <!--list products-->
    <th align="center">
      <xsl:for-each select="//product">
       <td><b><xsl:value-of select="."/></b></td>
      </xsl:for-each></th>
                                                <!--list customers-->
    <xsl:for-each select="/sales/record/cust">
     <xsl:variable name="customer" select="."/>
     <tr align="right"><td><xsl:value-of select="@num"/></td>
      <xsl:for-each select="//product">  <!--each product-->
       <td><xsl:value-of select="$customer/prodsale
                          [@idref=current()/@id]"/>
       </td></xsl:for-each>
     </tr></xsl:for-each>
                                                <!--summarize-->
    <tr align="right"><td><b>Totals:</b></td>
      <xsl:for-each select="//product">
       <xsl:variable name="pid" select="@id"/>
       <td><i><xsl:value-of
            select="sum(//prodsale[@idref=$pid])"/></i>
       </td></xsl:for-each></tr>
  </table>
 </body></html>
```

Product Rendering Results

Product Sales Summary

	Packing Boxes	Packing Tape
C1001	100	200
C1002		50
C1003	75	15
Totals:	*175*	*265*

Record of Sales

- C1001 - Packing Boxes - 100
- C1001 - Packing Tape - 200
- C1002 - Packing Tape - 50
- C1003 - Packing Boxes - 75
- C1003 - Packing Tape - 15