Applied Databases

Lecture 3
DTDs (regular expressions) & DOM

Sebastian Maneth

University of Edinburgh - January 18th, 2016

Outline

1. DTD Regular Expression → Glushkov Automaton
2. DOM Document Object Model

1. Regular Expressions

```
<!DOCTYPE addressbook [
    <!ELEMENT addressbook (person*) >
    <!ELEMENT person (name,greet*,address*,(fax|tel)*,email*)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT greet (#PCDATA)>
    <!ELEMENT address (#PCDATA)>
    <!ELEMENT fax (#PCDATA)>
    <!ELEMENT tel (#PCDATA)>
    <!ELEMENT email (#PCDATA)>
]>
```

1. Regular Expressions

- choice: `( .. | .. | .. )`

- sequence: `( .., .., .. )`

- optional: `...?`

- zero or more: `...*`

- one or more: `...+`

- element names

Note

→ #PCDATA may **not** appear in these regular expressions!

→ use *mixed content* instead

Regular Expressions are a very useful concept.

→ used in EBNF, for defining the syntax of PLs

→ used in various unix tools (e.g., `grep`)

→ supported in most PLs (esp. `Perl`), text editors

→ classical concept in CS (Stephen Kleene, 1950's)

How can you **implement** a regular expression?

Input: RegEx e, string w
Question: Does w *match* e?

→ use Finite Automata (FA)

Example
e = (ab | b)* a* a

w = a b b a a b a

match?

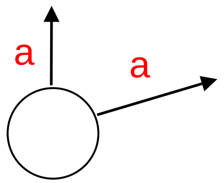
Finite-State Automata (FA)

→ *constant memory* computation

→ as Turing Machines, but *read-only* and *one-way* (left-to-right)

for every ReEx there is a FA (and vice versa)

Deterministic FA (DFA) = **no** two outgoing edges with same label



not
allowed

DFA Matching: time O(|DFA| + |w|)
"one finger needed"

FA Matching: time O(|FA| * |w|)
"at most #states many fingers needed"

→ every FA can be effectively transformed into an equivalent DFA.

→ can take exponential time! (“*subset construction*”)

How can you **implement** a regular expression?

Input: RegEx e, string w
Question: Does w *match* e?

```
FA = BuildFA(e);
FA.run;
→ or
DFA = BuildDFA(FA);
DFA.run
```

Running time O(|w| + 2^m)

or O(|w| * m)

To avoid these expensive running times

W3C requires that `BuildFA(e)` must be deterministic!

Is small! ☺
size is only O(m^2)

Running time O(n + mk)

W3C DTD-definition

Regular expressions e for which `BuildFA(e)`

is deterministic are called deterministic regular expressions.

→ max number of transitions (edges) for m states and k symbols?

To avoid these expensive running times

W3C requires that `BuildFA(e)` must be deterministic!

Is small! ☺
size is only O(m^2)

W3C DTD-definition

Running time O(n + mk)

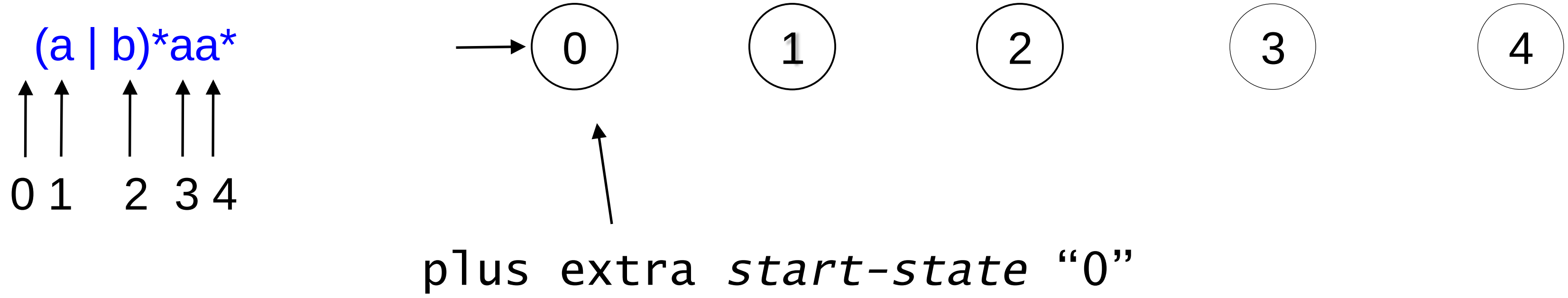
Regular expressions e for which `BuildFA(e)`

is deterministic are called deterministic regular expressions.

→ max number of transitions (edges) for m states and k symbols?

BuildFA(e) = “Glushkov Automaton” = “Position Automaton” [Glushkov1961]

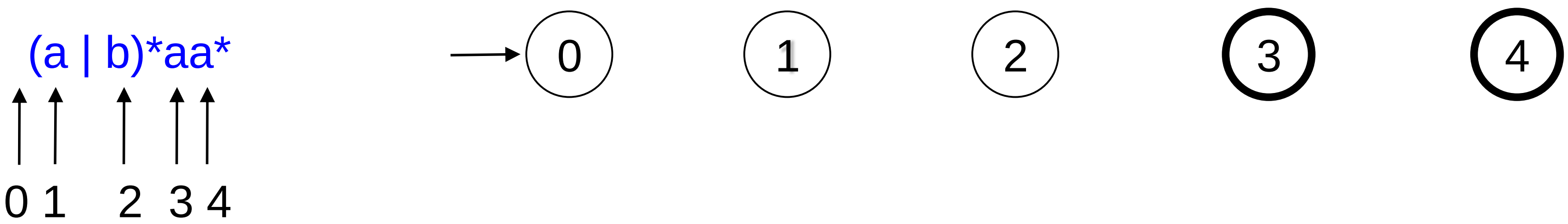
BuildFA(e) = every letter in e becomes a *state*



BuildFA(e) = “Glushkov Automaton” = “Position Automaton” [Glushkov1961]

→ identify **end-position(s)**

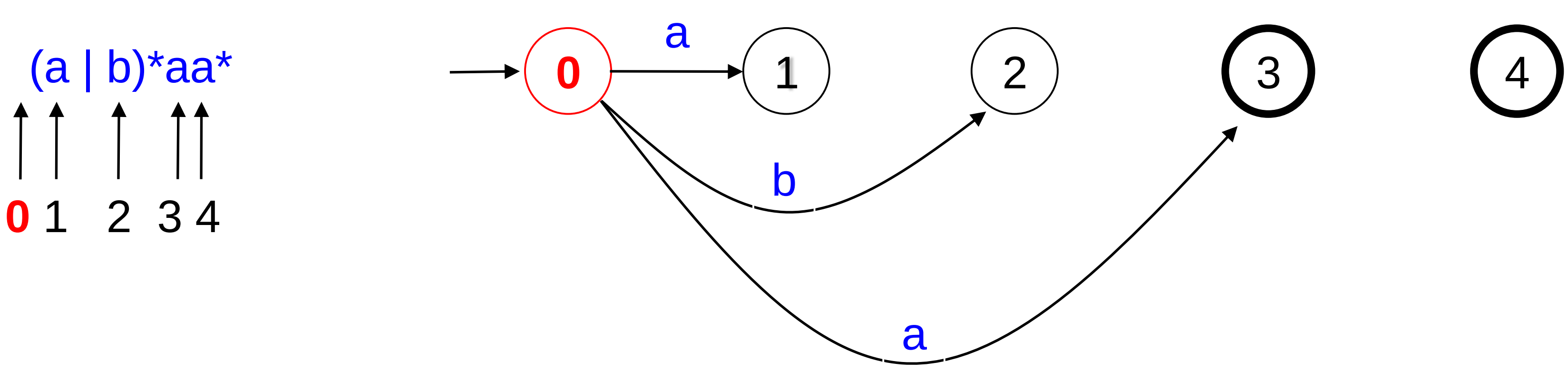
BuildFA(e) = every letter in e becomes a state



`BuildFA(e)` = “Glushkov Automaton” = “Position Automaton” [Glushkov1961]

→ which positions are reachable from “position 0”?

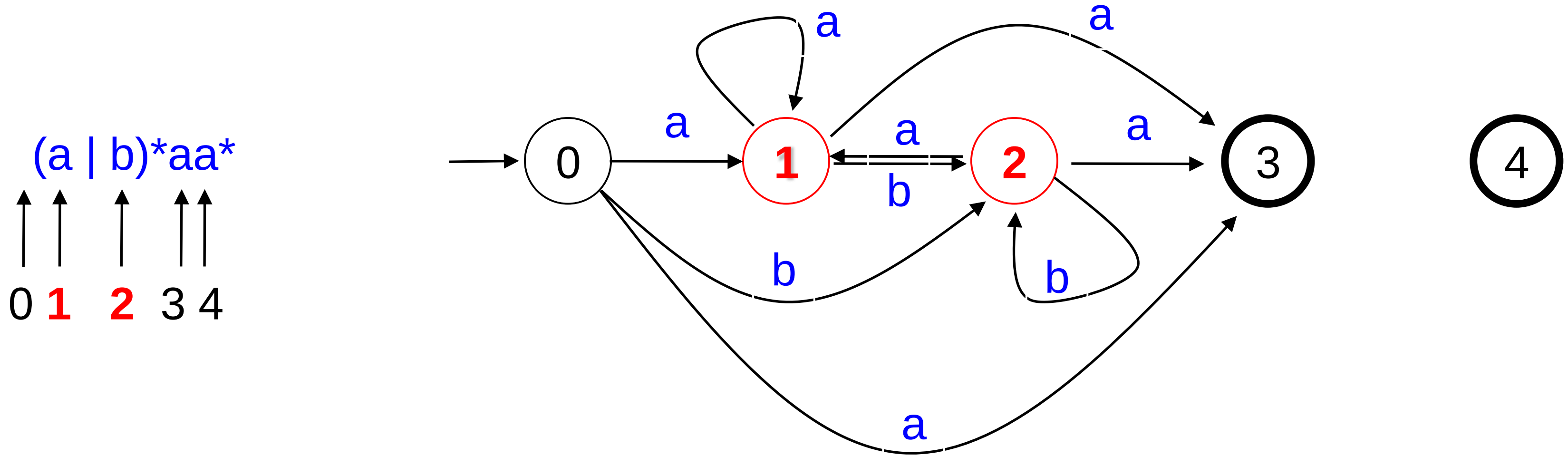
`BuildFA(e) = every letter in e becomes a state`



`BuildFA(e)` = “Glushkov Automaton” = “Position Automaton” [Glushkov1961]

→ which positions are reachable from positions 1 and 2?

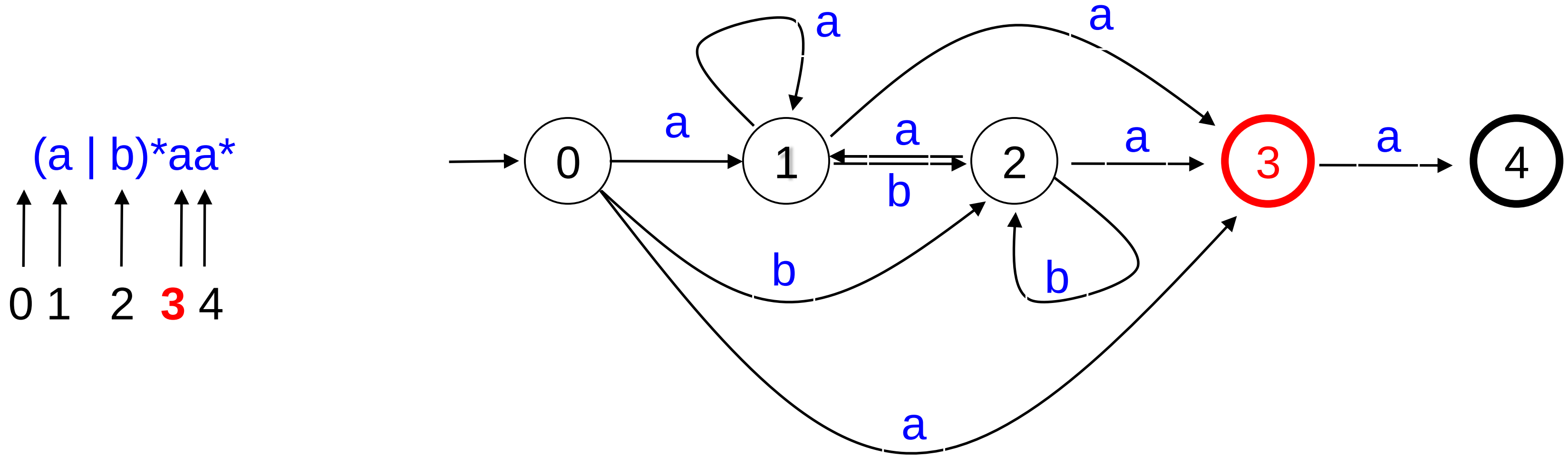
`BuildFA(e) = every letter in e becomes a state`



`BuildFA(e)` = “Glushkov Automaton” = “Position Automaton” [Glushkov1961]

→ which positions are reachable from position 3?

`BuildFA(e) = every letter in e becomes a state`

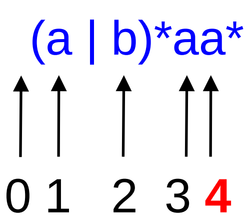


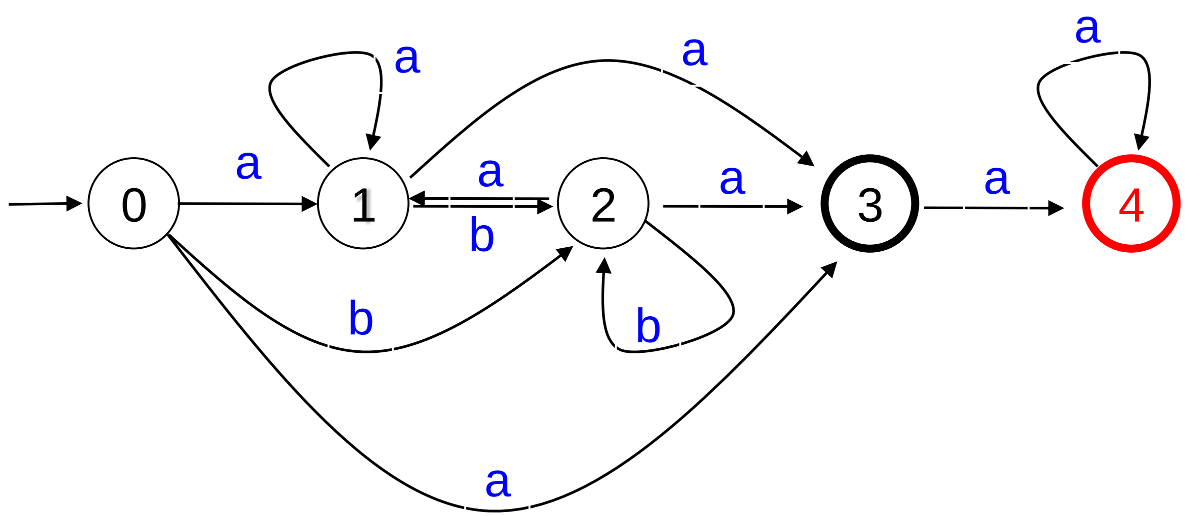
`BuildFA(e)` = “Glushkov Automaton” = “Position Automaton” [Glushkov1961]

→ which positions are reachable from position 4?

→ finished!

`BuildFA(e) = every letter in e becomes a state`

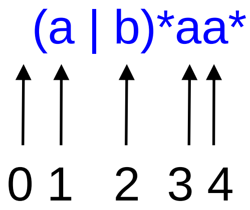


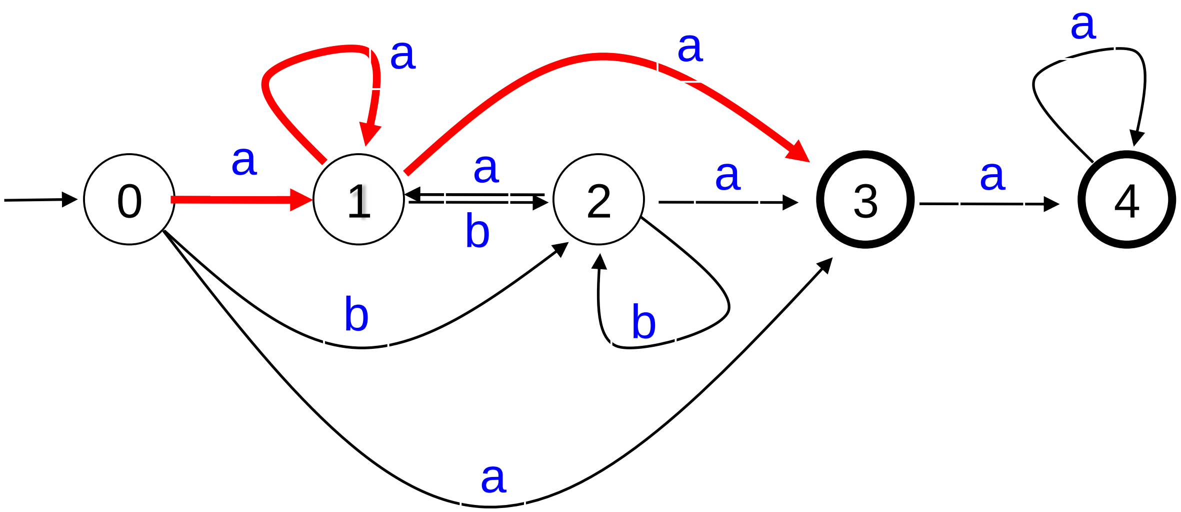


`BuildFA(e)` = "Glushkov Automaton" = "Position Automaton" [Glushkov1961]

→ a "**successful run**" for the input word "aaa"

`BuildFA(e) = every letter in e becomes a state`



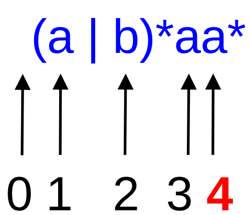


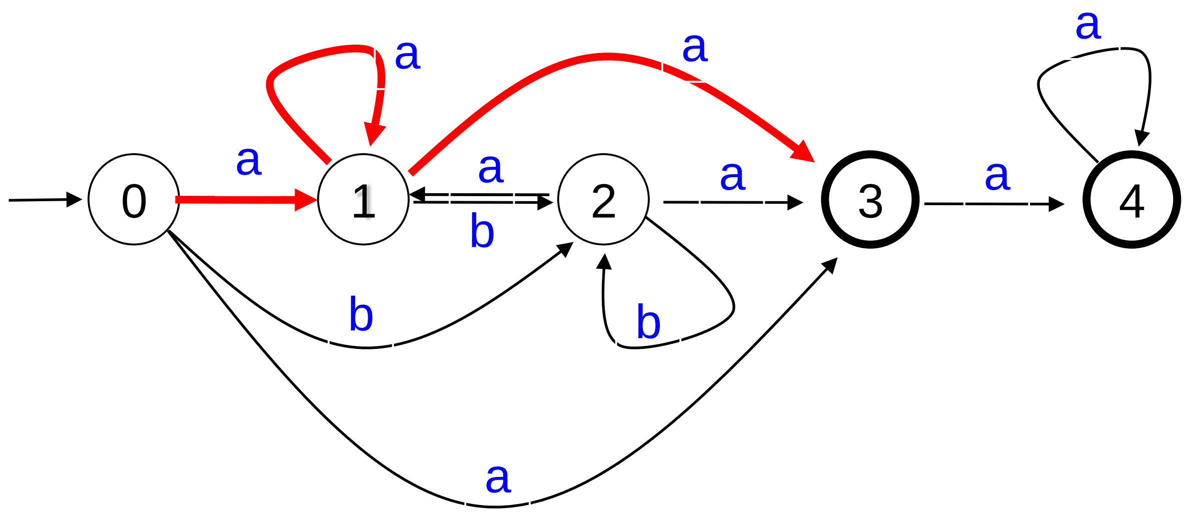
BuildFA(e) = “Glushkov Automaton” = “Position Automaton” [Glushkov1961]

→ a “**successful run**” for the input word “aaa”

→ how many other successful runs are there for “aaa”?

`BuildFA(e) = every letter in e becomes a state`





Document Type Definitions (DTDs)

- The XML specification restricts regular expressions in DTDs to be deterministic (***1-unambiguous***).

- Unambiguous regular expression: "*each word is witnessed by at most one sequence of positions of symbols in the expression that matches the word*" .[Brüggemann-Klein, Wood 1998]

  - ✓ Ambiguous expression (a | b)*aa* —mark with subscripts→ $(a_1 \mid b_1)^* a_2 a_3^*$
  - ✓ For aaa → three witnesses: $a_1a_1a_2$ $a_1a_2a_3$ $a_2a_3a_3$
  - ✓ Unambiguous equivalent expression : (a | b)*a

Document Type Definitions (DTDs)

- The XML specification restricts regular expressions in DTDs to be deterministic (***1-unambiguous***).

- Unambiguous regular expression: "*each word is witnessed by at most one sequence of positions of symbols in the expression that matches the word" .*[Brüggemann-Klein, Wood 1998]

  - ✓ Ambiguous expression $(a \mid b)^*aa^*$ —mark with subscripts→ $(a_1 \mid b_1)^*a_2a_3^*$
  - ✓ For aaa → three witnesses: $a_1a_1a_2$ $a_1a_2a_3$ $a_2a_3a_3$
  - ✓ Unambiguous equivalent expression : $(a \mid b)^*a$ ***not*** 1-unambiguous!

→ 1-unambiguous: decide position by looking **only at current symbol**
consider baa: b_1a**?**

Questions for each expression, deterministic or not?

→ a?b?

→ a?b?a

→ a(aba)*b

→ (a?b?c?d?e?)*

Questions for each expression, deterministic or not?

→ a?b?

→ a?b?a

→ a(aba)*b

→ (a?b?c?d?e?)* ← How many edges in the Glushkov automaton?
(a1?a2? …. ak?) for distinct a1,a2,...

Questions for each expression, deterministic or not?

→ a?b?

→ a?b?a

→ a(aba)*b

→ (a?b?c?d?e?)* ← How many edges in the Glushkov automaton?

→ (a | b)*a is not deterministic.

Can you find an equivalent expression that **is** deterministic?

Questions for each expression, deterministic or not?

→ a?b?

→ a?b?a

→ a(aba)*b

→ (a?b?c?d?e?)* ← How many edges in the Glushkov automaton?

→ (a | b)*a is not deterministic.

Can you find an equivalent expression that **is** deterministic?

→ (a | b)*a(a | b) is not deterministic.

Can you find an equivalent expression that **is** deterministic?

Questions for each expression, deterministic or not?

→ a?b?

→ a?b?a

→ a(aba)*b

→ (a?b?c?d?e?)* ← How many edges in the Glushkov automaton?

Notes

→ there exist regular expressions for which no equivalent deterministic expressions exist

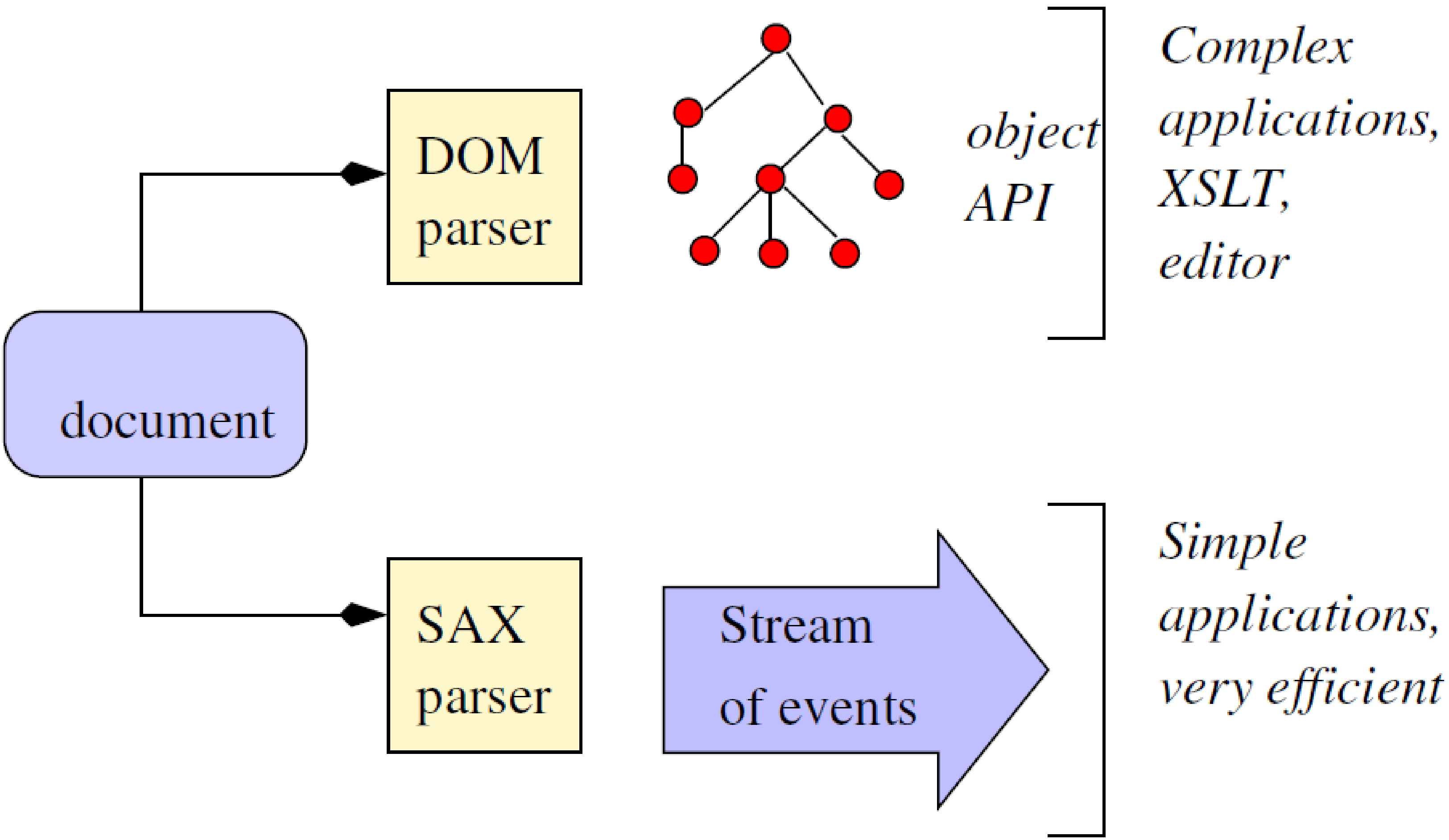
→ this can be decided, and an equivalent deterministic reg expr constructed if it exists [Brüggemann-Klein, Wood 1998]

XML Parsers

→ Document Object Model - DOM

→ Simple API for XML - SAX

XML Parsers


DOM
parser
object
API
Complex
applications,
XSLT,
editor
document
SAX
parser
Stream
of events
Simple
applications,
very efficient

XML Parsers

→ DOM - loads full document into memory

→ SAX - generates streaming events
- nothing stored in memory

2. DOM – Document Object Model

→ Language and platform-independent view of XML

→ DOM APIs exist for many PLs (Java, C++, C, Perl, Python, …)

DOM relies on two main concepts

(1) The XML processor constructs the
complete XML document tree (in-memory)

(2) The XML application issues DOM library calls to **explore** and **manipulate** the XML tree, or to **generate** new XML trees.

Advantages
- easy to use
- once in memory, no tricky issues with XML syntax anymore
- all DOM trees serialize to well-formed XML (even after arbitrary updates)!

2. DOM – Document Object Model

→ Language and platform-independent view of XML

→ DOM APIs exist for many PLs (Java, C++, C, Perl, Python, …)

DOM relies on two main concepts

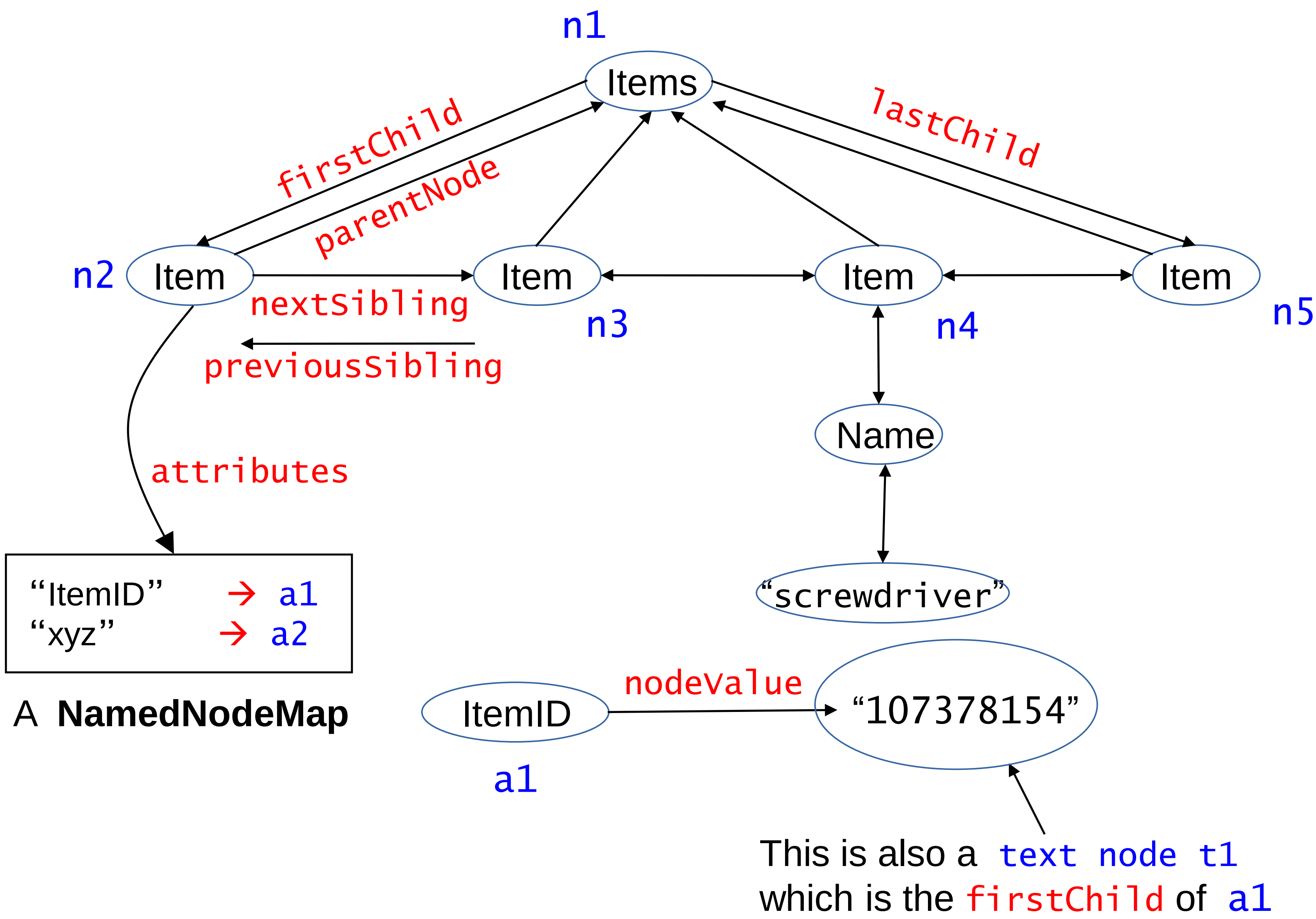
(1) The XML processor constructs the **complete XML document tree** (in-memory)

(2) The XML application issues DOM library calls to **explore** and **manipulate** the XML tree, or to **generate** new XML trees.

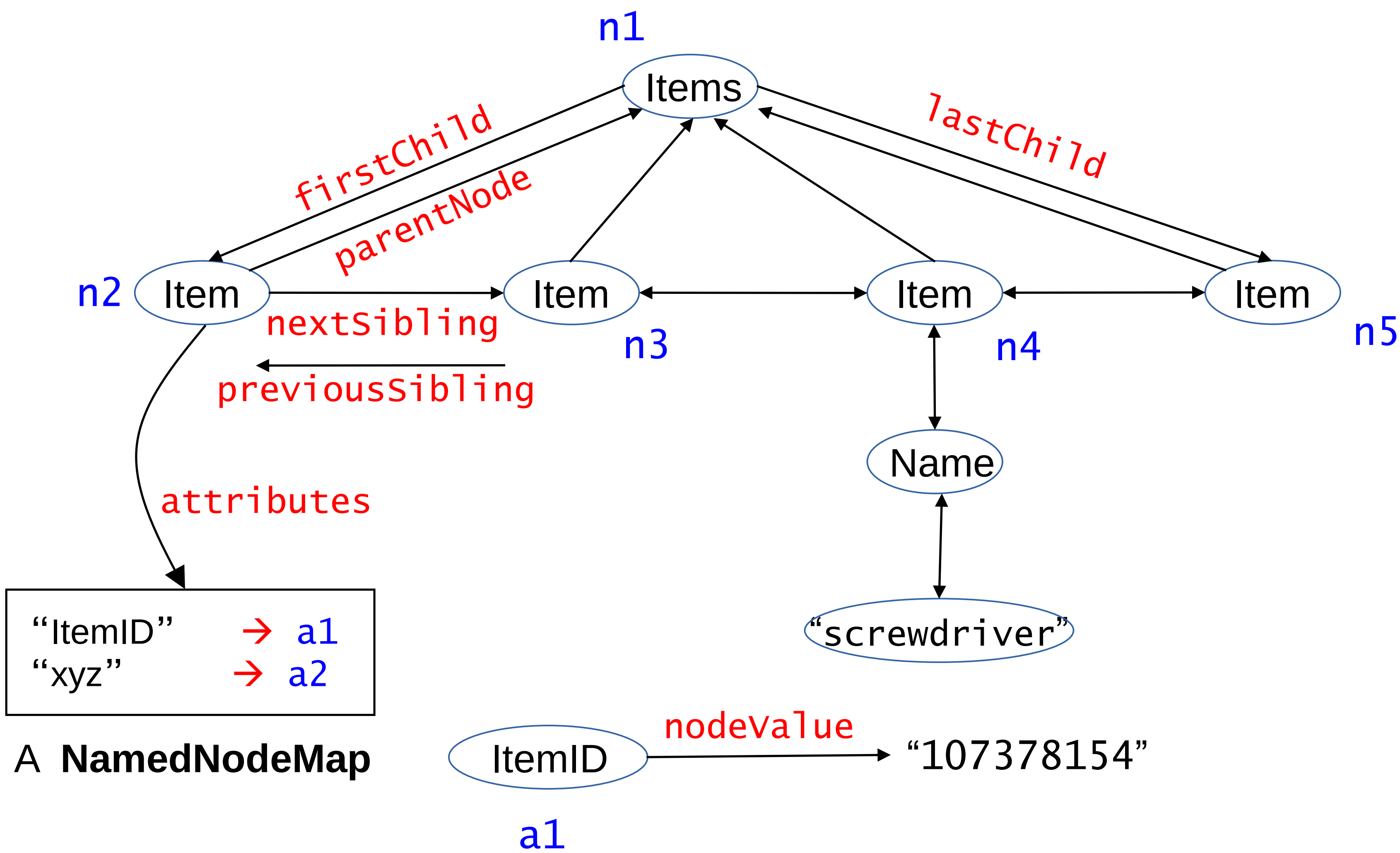
Advantages
- easy to use
- once in memory, no tricky issues with XML syntax anymore
- all DOM trees serialize to well-fromed XML (even after arbitrary updates)!

Disadvantage Uses LOTS of memory!!

2. DOM – Document Object Model



2. DOM – Document Object Model



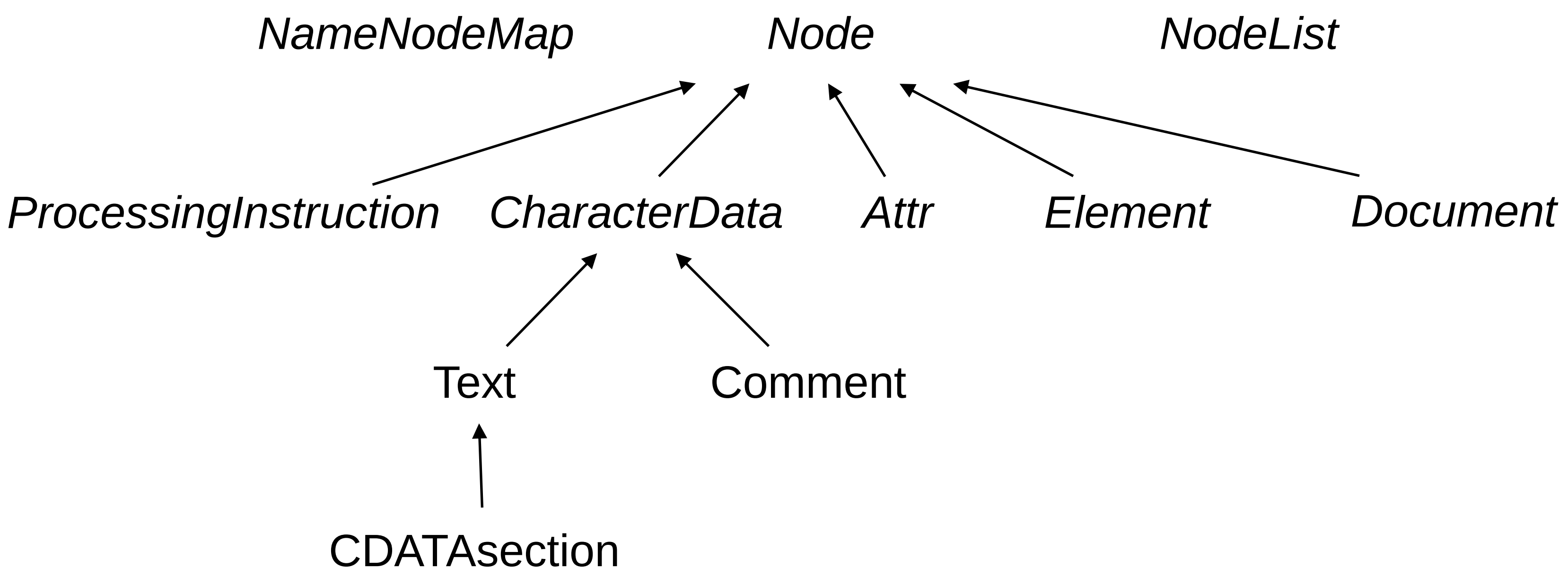
→ how much memory is needed for a typical XML document of 1GB?

DOM Level 1 (Core)

Some methods

DOM type	Method	Comment
Node	nodeName, nodeValue : DOMString	redefined in subclasses
	parentNode : Node	
	firstChild : Node	leftmost child
	lastChild : Node	rightmost child
	nextSibling : Node	returns NULL for root elem or last child or attributes
	previousSibling : Node	
	childNodes : NodeList	
	attributes : NamedNodeMap	
	ownerDocument: Document	
	replaceChild : Node	
Document	createElement : Element	creates element with given tag name
	createComment : Comment	
	getElementsByTagName: NodeList	list of all Elem nodes in document order

DOM Level 1 (Core)



Character strings (DOM type *DOMString*) are defined to be encoded using UTF-16 (e.g., Java DOM reresents type *DOMString* using its `String` type).

DOM Level 1 (Core)

Name,Value, and attributes depend on the **type** of the current node.

The values of `nodeName`, `nodeValue`, and `attributes` vary according to the node type as follows:

	nodeName	**nodeValue**	**attributes**
Element	tagName	null	NamedNodeMap
Attr	name of attribute	value of attribute	null
Text	#text	content of the text node	null
CDATASection	#cdata-section	content of the CDATA Section	null
EntityReference	name of entity referenced	null	null
Entity	entity name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Comment	#comment	content of the comment	null
Document	#document	null	null
DocumentType	document type name	null	null
DocumentFragment	#document-fragment	null	null
Notation	notation name	null	null

DOM Level 1 (Core)

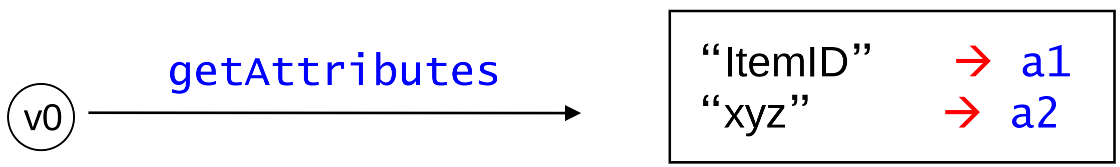
Some details

Creating an *element/attribute* using *createElement/createAttribute* does not wire the new node with the XML tree structure yet.
→Call *insertBefore, replaceChild*, …, to wire a node at an explicity position

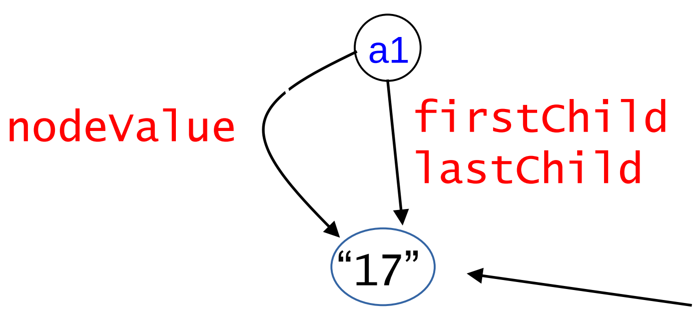
DOM type *NodeList* makes up fo the lack of collection data types in many programming languages

DOM type *NamedNodeMap* represents an *association table* (nodes may be accessed by name)

Example:



A **NamedNodeMap**



Methods: *getNamedItem*, *setNamedItem*,…

text node! (with nodeValue="17")

```java
public static void recursiveDescent(Node n, int level) {

    // adjust indentation according to level
    for(int i=0; i<4*level; i++) System.out.print("    ");

    // dump out node name, type, and value
    String ntype = typeName[n.getNodeType()];
    String nname = n.getNodeName();
    String nvalue = n.getNodeValue();
    System.out.println("Type = " + ntype + ", Name = " + nname + ",
                        Value = " + nvalue);

    // dump out attributes if any
    org.w3c.dom.NamedNodeMap nattrib = n.getAttributes();
    if(nattrib != null && nattrib.getLength() > 0)
        for(int i=0; i<nattrib.getLength(); i++)
            recursiveDescent(nattrib.item(i), level+1);

    // now walk through children list
    org.w3c.dom.NodeList nlist = n.getChildNodes();
    for(int i=0; i<nlist.getLength(); i++)
        recursiveDescent(nlist.item(i), level+1);
}
```

```
<?xml version="1.0"?>
<!DOCTYPE greeting [
  <!ENTITY hi "Hello">
  <!ENTITY hi1 "&hi;&hi;">
  <!ENTITY hi2 "&hi1;&hi1;">
  <!ENTITY hi3 "&hi2;&hi2;">
  <!ENTITY s "<d></d>">
]>
<a a1='17' a2='29'><b>xy &hi3; world &s; zz</b></a>
```

file.xml

element node!

```
$ java MyDOM file.xml
Successfully parsed - file.xml
Type = Document, Name = #document, Value = null
    Type = DocType, Name = greeting, Value = null
    Type = Element, Name = a, Value = null
        Type = Attr, Name = a1, Value = 17
            Type = Text, Name = #text, Value = 17
        Type = Attr, Name = a2, Value = 29
            Type = Text, Name = #text, Value = 29
        Type = Element, Name = b, Value = null
            Type = Text, Name = #text, Value = xy HelloHelloHelloH
elloHelloHelloHelloHello world
            Type = Element, Name = d, Value = null
            Type = Text, Name = #text, Value =  zz
```

END
Lecture 3