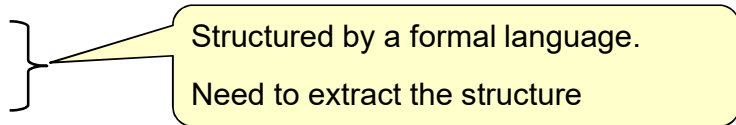

COMP 261 2024 Tri 1

Parsing

Dealing with Text Data

- COMP 102 & 103 assumed data files were structured very simply:
 - a) Sequence of Strings, one string per line.
 - b) Sequence of "tokens" (numbers or words), space-separated
 - c) Sequence of "tabular" lines:
each line is a sequence of space-separated tokens of a fixed pattern.
- Used a simple Scanner to separate the words and numbers.
- Real-world text data is not so simple!
 - Text with Markup (eg HTML, XML)
 - Programs written in programming languages
 - Semi-structured text data with messy formatting.
 - Natural language text is complicated (paragraphs, sentences, punctuation, parts-of-speech, grammatical structure, footnotes, word endings, words with meaning.....)



Structured by a formal language.

Need to extract the structure

Examples of formal languages

XML documents:

```
<html><head><title>My Web Page</title></head>  
<body><p> Thanks for viewing </p></body></html>
```

Java statement:

```
while (A[k]!=x){if (k>0){A[k]+=A[k-1];} k++}
```

SQL schema definition or query:

```
DELETE FROM DomesticStudentsFor2022  
WHERE mark = 'E';
```

Examples of formal languages

latex document:

```
\subquestion[7] Work out the Big-0 cost of the \cd{changeList(..)} method below:  
\par\vspace{-0.4em}\begin{itemize}\setlength{\itemsep}{0.0em}  
\item working out the cost of performing each line once.  
\item working out the number of times each line will be performed.  
\item computing the total cost, giving just the most significant term.  
\end{itemize}  
Assume the size of the list in \cd{data} is \mathit{(n)}.  
\par\medskip  
\begin{answerCode}\begin{lstlisting}[style=jjcode]
```

Examples of formal languages

foswiki document:

```
---+ Schedule
```

```
---++ %COURSECODE% : Schedule of lectures, tutorials, assignments, tests, and holidays.
```

See the `[[TimeTable]]` for times and locations of lectures and labs

Video recordings of lectures will be available from Panopto, which is accessible through `[[https://nuku.wgtn.ac.nz/courses/12324][Nuku/Canvas]]`.

```
<!-- ----- -->
```

```
| *Week 1: 27 Feb - 5 March* ||
```

```
| Lectures | [[[%ATTACHURL%/week1.pdf][slides]] |
```

```
| [[Assignment1][Assignment 1]] out | Parsing and interpreting a language |
```

```
| *Week 2: 6-12 March* ||
```

```
| Lectures | [[[%ATTACHURL%/week2.pdf][slides]] |
```

```
| Fri 10 Mar | Deadline for withdrawing with fee refund |
```

```
| *Week 3: 13 - 19 March* ||
```

A "robot" control program for a game (Assignment 1)

```
move(8);
turnL;
loop {
    while ( or(eq(numBarrels, 0), lt(add(oppFB, oppLR), add(barrelFB,barrelLR))) ) {
        if (lt(oppFB,0)) { turnAround; }
        else { if (gt(oppFB,0)) { move(add(1, div(oppFB, 2))); }
        else { if (lt(oppLR,0)) { turnL;}
        else { if (gt(oppLR,0)) { turnR;}
        else { if (eq(oppLR,0)) { takeFuel; }
        }}}
    }
    if ( and(eq(barrelFB, 0),eq(barrelLR, 0))) { takeFuel; }
    else { if ( lt(barrelFB, 0) ){ turnAround; }
    else { if ( gt(barrelFB, 0) ) { move(barrelFB); }
    else { if ( lt(barrelLR, 0) ) { turnL; }
    else { if ( gt(barrelLR, 0) ) { turnR; }
    }}}
}
```

Formal Languages

- Formal languages can be described by a **grammar** – a set of rules describing how to construct valid examples of the language
- The **grammar rules** specify the structure of the text in that language.
- A **parser** uses the grammar rules to parse a program/web page/marked-up document, and construct a data-structure containing the structure of the text.

Questions:

- How do we write the grammar rules for a language?
- How do we represent the structure?
- How can we construct a parser for a language?

Assignment: parser for a programming language for a robot game.

A grammar example

A simple html grammar:

```
HTMLFILE ::= "<html>" [ HEAD ] BODY "</html>"
HEAD ::= "<head>" TITLE "</head>"
TITLE ::= "<title>" TEXT "</title>"
BODY ::= "<body>" [ BODYTAG ]* "</body>"
BODYTAG ::= H1TAG | PTAG | OLTAG | ULTAG
H1TAG ::= "<h1>" TEXT "</h1>"
PTAG ::= "<p>" TEXT "</p>"
OLTAG ::= "<ol>" [ LITAG ]+ "</ol>"
ULTAG ::= "<ul>" [ LITAG ]+ "</ul>"
LITAG ::= "<li>" TEXT "</li>"
TEXT ::= sequence of characters other than < and >
```


The grammar of grammars

A simple html grammar:

```

HTMLFILE ::= "<html>" [ HEAD ] BODY "</html>"
HEAD ::= "<head>" TITLE "</head>"
TITLE ::= "<title>" TEXT "</title>"
BODY ::= "<body>" [ BODYTAG ]* "</body>"
BODYTAG ::= H1TAG | PTAG | OLTAG | ULTAG
H1TAG ::= "<h1>" TEXT "</h1>"
PTAG ::= "<p>" TEXT "</p>"
OLTAG ::= "<ol>" [ LITAG ]+ "</ol>"
ULTAG ::= "<ul>" [ LITAG ]+ "</ul>"
LITAG ::= "<li>" TEXT "</li>"
TEXT ::= sequence of characters other than < and >

```

::= = a grammar rule
 | = "or"
 [...] = "optional"
 [...] * = "any number
 of times"
 [...] + = "one or more
 times"

Grammars: Nonterminals

Nonterminals

- elements of the grammar, specifying structure, that are not part of the text
- defined by rules.

HTMLFILE ::= "<html>" [**HEAD**] **BODY** "</html>"

HEAD ::= "<head>" **TITLE** "</head>"

TITLE ::= "<title>" **TEXT** "</title>"

BODY ::= "<body>" [**BODYTAG**]* "</body>"

BODYTAG ::= **H1TAG** | **PTAG** | **OLTAG** | **ULTAG**

H1TAG ::= "<h1>" **TEXT** "</h1>"

PTAG ::= "<p>" **TEXT** "</p>"

OLTAG ::= "" [**LITAG**]+ ""

ULTAG ::= "" [**LITAG**]+ ""

LITAG ::= "" **TEXT** ""

TEXT ::= *sequence of characters other than < and >*

Top level
nonterminal
usually first

Grammars: Terminals

Terminals

- literal strings or patterns of characters that will match bits of the text.

HTMLFILE ::= “<html>” [HEAD] BODY “</html>”

HEAD ::= “<head>” TITLE “</head>”

TITLE ::= “<title>” TEXT “</title>”

BODY ::= “<body>” [BODYTAG]* “</body>”

BODYTAG ::= H1TAG | PTAG | OLTAG | ULTAG

H1TAG ::= “<h1>” TEXT “</h1>”

PTAG ::= “<p>” TEXT “</p>”

OLTAG ::= “” [LITAG]+ “”

ULTAG ::= “” [LITAG]+ “”

LITAG ::= “” TEXT “”

TEXT ::= **sequence of characters other than < and >**

Using the Grammar

Given some text:

```
<html>
<head><title> Today</title></head>
<body><h1> My Day </h1>
<ul><li>meeting</li><li> lecture </li></ul>
<p> parsing stuff</p>
</body>
</html>
```

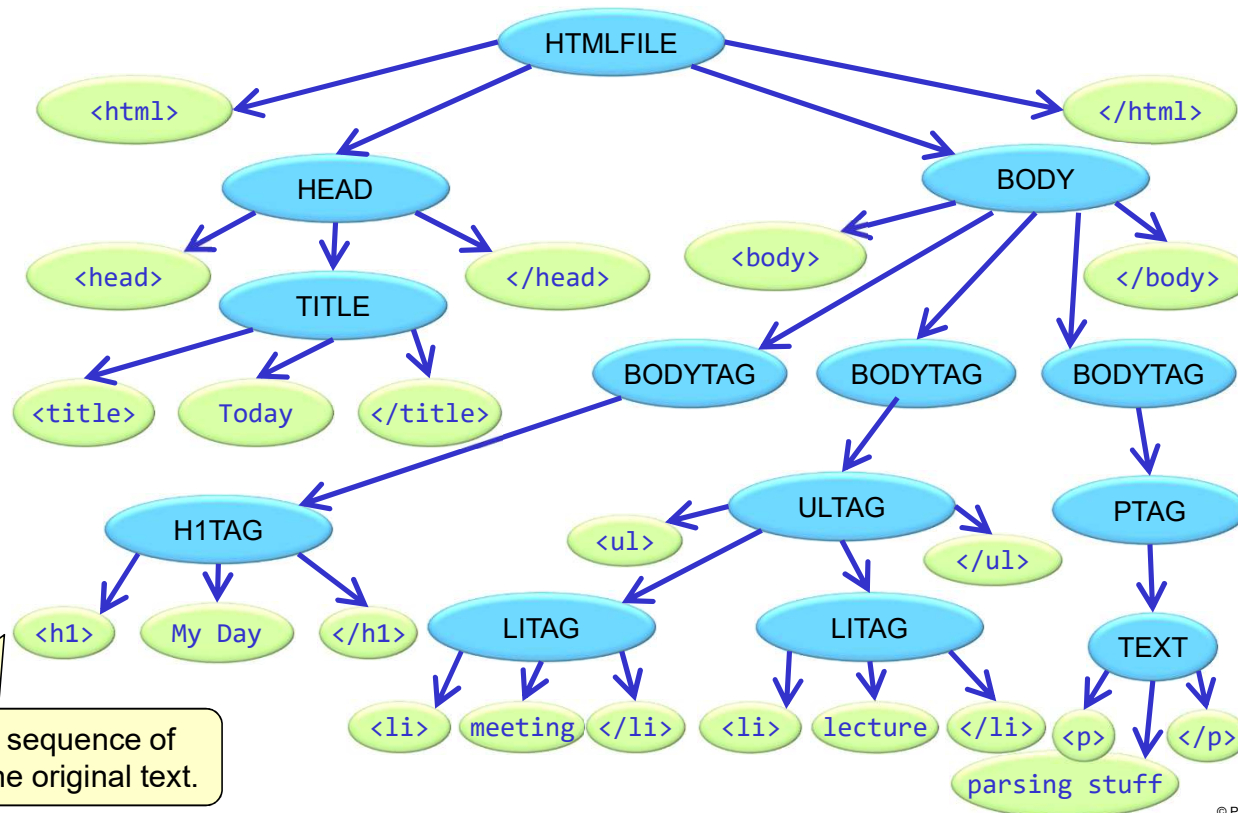
- Question 1: Is it a valid piece of HTML?
 - Does it conform to the grammar rules?
- Question 2: What is the structure of the text? (Needed in order to process it)
 - what are the components?
 - what types are the components?
 - how are they related?

What kind of structure?

- Text conforming to a grammar has a tree structure
 - Ordered tree – order of the children matters
 - Each node in the tree and its children correspond to a grammar rule
 - Each internal node labeled by the nonterminal on LHS of the rule
 - Leaves correspond to terminals.

Concrete Parse Tree:

COMP261 # 14



Depth-first sequence of leaves = the original text.

© Peter Andreae and Xiaoying Gao

Concrete parse tree has too much information.

To render the web page, we don't need all that information!

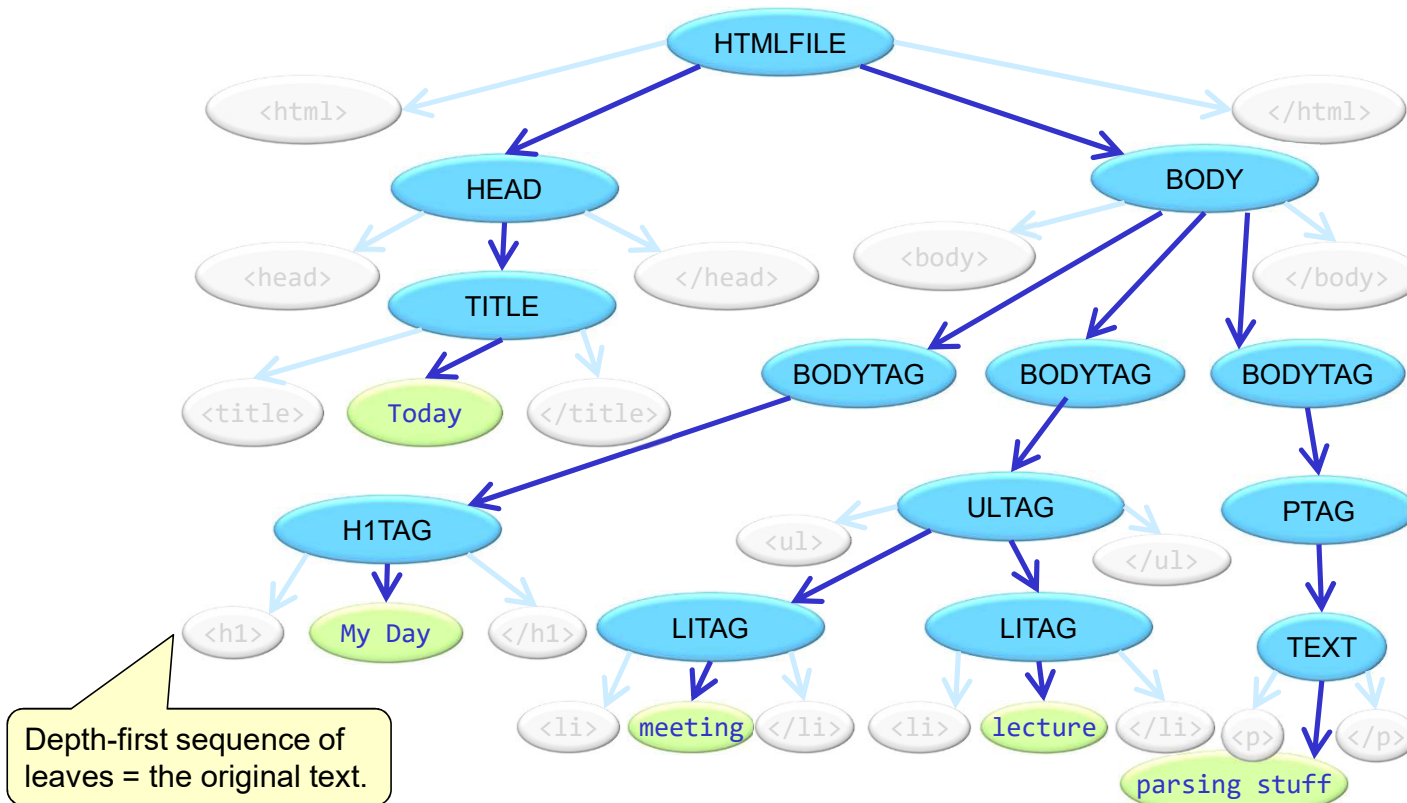
- For example, we know that every HEAD will contain “<head>” and “</head>” terminals, we only care about what TITLE there is and only the unknown string part of that title.

Definition:

1. An **abstract syntax tree (AST)** is a tree representation of the abstract syntactic structure of the text.
2. Each node of the tree denotes a construct occurring in the text.
3. The syntax is ‘abstract’ in that it does not represent all the elements of the full syntax.

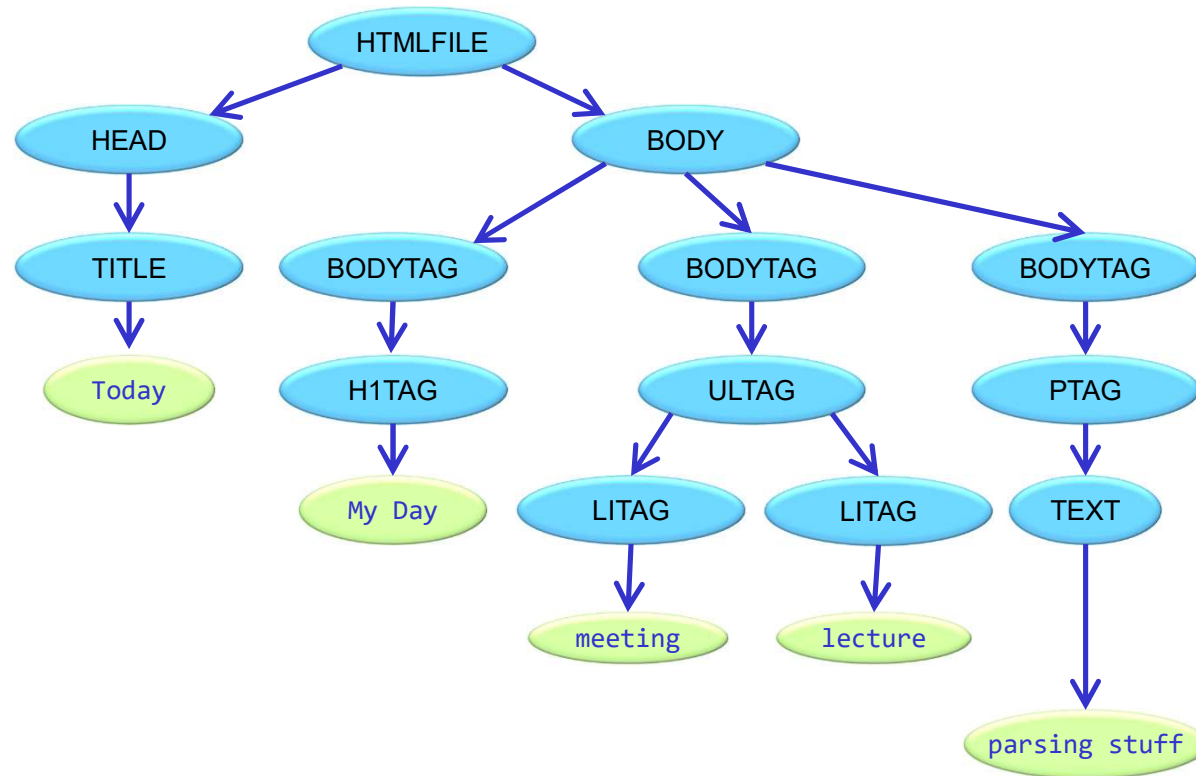
Abstract Syntax Tree:

COMP261 # 16



Abstract Syntax Tree (AST)

COMP261 # 17



© Peter Andreae and Xiaoying Gao