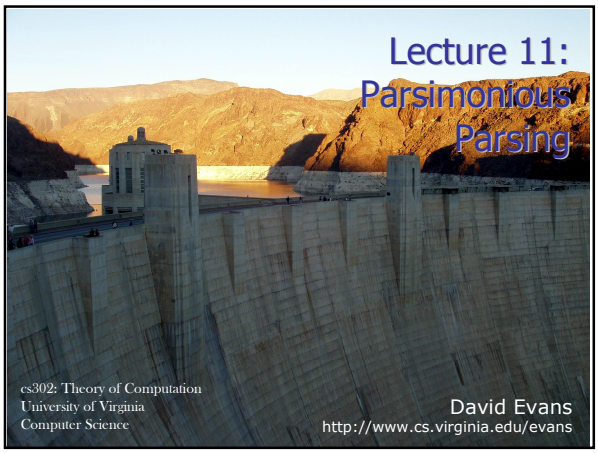


Menu

- Fix proof from last class
- Interpretive Dance!
- Parsimonious Parsing (Parsimoniously)

PS3 Comments Available Today
PS3 will be returned Tuesday

Closure Properties of CFLs

If A and B are *context free* languages then:

A^R is a context-free language **TRUE**

A^* is a context-free language **TRUE**

$\overline{A}$ is a context-free language (complement)?

$A \cup B$ is a context-free language **TRUE**

$A \cap B$ is a context-free language ?

Complementing Non-CFLs

$L_{ww} = \{ ww \mid w \in \Sigma^* \}$ is **not** a CFL.
Is its complement?

Yes. This CFG recognizes is:

What is the actual language?

$S \rightarrow \mathbf{0}S\mathbf{0} \mid \mathbf{1}S\mathbf{1} \mid \mathbf{0}X\mathbf{1} \mid \mathbf{1}X\mathbf{0}$

$X \rightarrow \mathbf{0}X\mathbf{0} \mid \mathbf{1}X\mathbf{1} \mid \mathbf{0}X\mathbf{1} \mid \mathbf{1}X\mathbf{0} \mid \mathbf{0} \mid \mathbf{1} \mid \varepsilon$

Bogus Proof!

$S \rightarrow \mathbf{0}X\mathbf{1} \rightarrow \mathbf{01}X\mathbf{01} \rightarrow \mathbf{0101} \in L_{ww}$

CFG for $\overline{L_{ww}}$ $(L_{\neg ww})$

All odd length strings are in $L_{\neg ww}$

$S \rightarrow S_{\text{Odd}} \mid S_{\text{Even}}$

$S_{\text{Odd}} \rightarrow \mathbf{0}R \mid \mathbf{1}R \mid \mathbf{0} \mid \mathbf{1}$
$R \rightarrow \mathbf{0}S_{\text{Odd}} \mid \mathbf{1}S_{\text{Odd}}$

$S_{\text{Even}} \rightarrow XY \mid YX$
$X \rightarrow ZXZ \mid \mathbf{0}$
$Y \rightarrow ZYZ \mid \mathbf{1}$
$Z \rightarrow \mathbf{0} \mid \mathbf{1}$

How can we prove this is correct?

S_{odd} generates all odd-length strings

$S_{\text{Odd}} \rightarrow \mathbf{0}R \mid \mathbf{1}R \mid \mathbf{0} \mid \mathbf{1}$
$R \rightarrow \mathbf{0}S_{\text{Odd}} \mid \mathbf{1}S_{\text{Odd}}$

Proof by induction on the length of the string.
Basis. S_{Odd} generates all odd-length strings of length 1. There are two possible strings: **0** and **1**. They are produces from the 3rd and 4th rules.

Induction. Assume S_{Odd} generates all odd-length strings of length n for $n = 2k+1$, $k \geq 0$. Show it can generate all odd-length string of length $n+2$.

S_{Odd} generates all odd-length strings

$S_{\text{Odd}} \to \mathbf{0}R \mid \mathbf{1}R \mid \mathbf{0} \mid \mathbf{1}$
$R \to \mathbf{0}S_{\text{Odd}} \mid \mathbf{1}S_{\text{Odd}}$

Induction. Assume S_{Odd} generates all odd-length strings of length n for $n = 2k+1$, $k \geq 0$. Show it can generate all odd-length string of length $n+2$.

All $n+2$ length strings are of the form abt where t is an n-length string and $a \in \{0, 1\}$, $b \in \{0, 1\}$. There is some derivation from $S_{\text{Odd}} \Rightarrow^* t$ (by the induction hypothesis). We can generate all four possibilities for a and b:

$\mathbf{00}t$: $S_{\text{Odd}} \to \mathbf{0}R \to \mathbf{00}S_{\text{Odd}} \Rightarrow^* \mathbf{00}t$
$\mathbf{01}t$: $S_{\text{Odd}} \to \mathbf{0}R \to \mathbf{01}S_{\text{Odd}} \Rightarrow^* \mathbf{01}t$
$\mathbf{10}t$: $S_{\text{Odd}} \to \mathbf{1}R \to \mathbf{10}S_{\text{Odd}} \Rightarrow^* \mathbf{10}t$
$\mathbf{11}t$: $S_{\text{Odd}} \to \mathbf{1}R \to \mathbf{11}S_{\text{Odd}} \Rightarrow^* \mathbf{01}t$

CFG for $\overline{L_{ww}}$ $(L_{\neg ww})$

$S \to S_{\text{Odd}} \mid S_{\text{Even}}$

$S_{\text{Odd}} \to \mathbf{0}R \mid \mathbf{1}R \mid \mathbf{0} \mid \mathbf{1}$
$R \to \mathbf{0}S_{\text{Odd}} \mid \mathbf{1}S_{\text{Odd}}$

$S_{\text{Even}} \to XY \mid YX$
$X \to ZXZ \mid \mathbf{0}$
$Y \to ZYZ \mid \mathbf{1}$
$Z \to \mathbf{0} \mid \mathbf{1}$

?

Proof-by-leaving-as-"Challenge Problem" (note: you cannot use this proof technique in your answers)

Even Strings

$S_{\text{Even}} \to XY \mid YX$
$X \to ZXZ \mid \mathbf{0}$
$Y \to ZYZ \mid \mathbf{1}$
$Z \to \mathbf{0} \mid \mathbf{1}$

Show S_{Even} generates the set of all even-length strings that are not in L_{ww}.

Proof by induction on the length of the string.
Basis. S_{Even} generates all even-length strings of length 0 that are not in L_{ww}. The only length 0 string is ε. ε is in L_{ww} since ε = εε, so ε should not be generated by S_{Even}. Since S_{Even} does not contain any right sides that go to ε, this is correct.

Closure Properties of CFLs

If A and B are *context free* languages then:

A^R is a context-free language **TRUE**

A^* is a context-free language **TRUE**

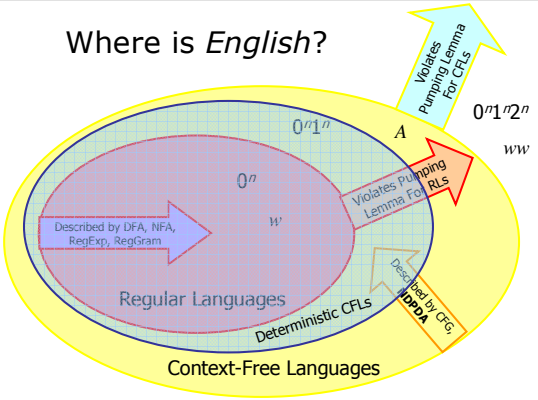
$\overline{A}$ is **not necessarily** a context-free language (complement)

$A \cup B$ is a context-free language **TRUE**

$A \cap B$ is a context-free language ?

Left for you to solve (possibly on Exam 1)

Where is *English*?



English ∉ Regular Languages

The cat likes fish.

The cat the dog chased likes fish.

The cat the dog the rat bit chased likes fish.

…

This is a pumping lemma proof!

LIMITATIONS OF
PHRASE STRUCTURE DESCRIPTION

5.1 We have discussed two models for the structure of language, a communication theoretic model based on a conception of language as a Markov process and corresponding, in a sense, to the minimal linguistic theory, and a phrase structure model based on immediate constituent analysis. We have seen that the first is surely inadequate for the purposes of grammar, and that the second is more powerful than the first, and does not fail in the same way. Of course there are languages (in our general sense) that cannot be described in terms of phrase structure, but I do not know whether or not English is itself literally outside the range of such analysis. However, I think that there are other grounds for rejecting the theory of phrase structure as inadequate for the purpose of linguistic description.

The strongest possible proof of the inadequacy of a linguistic theory is to show that it literally cannot apply to some natural language. A weaker, but perfectly sufficient demonstration of inadequacy would be to show that the theory can apply only clumsily; that is, to show that any grammar that can be constructed in terms of this theory will be extremely complex, *ad hoc*, and 'unrevealing', that certain very simple ways of describing grammatical sentences cannot be accommodated within the associated forms of grammar, and that certain fundamental formal properties of natural language cannot be utilized to simplify grammars. We can gather a good deal of evidence of this sort in favor of the thesis that the form of grammar described above, and the conception of linguistic theory that underlies it, are fundamentally inadequate.

The only way to test the adequacy of our present apparatus is to attempt to apply it directly to the description of English sentences.

= DFA
= CFG

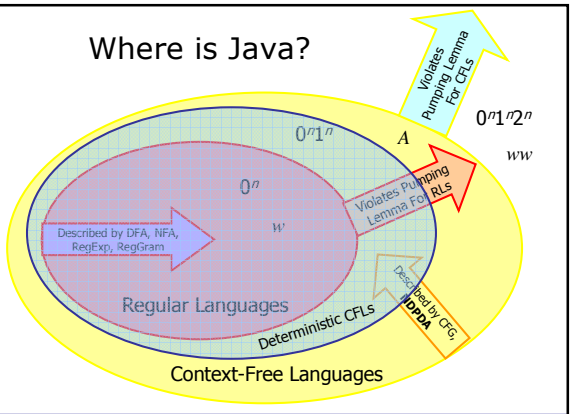
Chomsky's Answer (*Syntactic Structures*, 1957)

Current Answer

- Most linguists argue that most natural languages are not context-free
- But, it is hard to really answer this question:

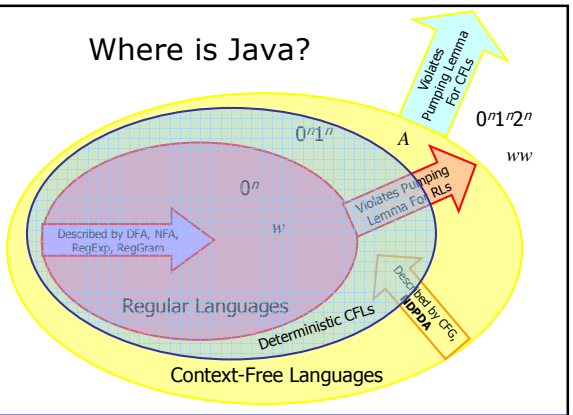
e.g.,
"The cat the dog the rat bit chased likes fish." ∈ English?

Where is Java?



Interpretive Dance

Where is Java?



What is the Java Language?

```
public class Test {
    public static void main(String [] a) {
        println("Hello World!");
    }
}
```

In the Java Language

```
Test.java:3: cannot resolve symbol
symbol  : method println  (java.lang.String)
```

```
// C:\users\luser\Test.java
public class Test {
    public static void main(String [] a) {
        println ("Hello Universe!");
    }
} }
```

Not in the Java Language

```
Test.java:1: illegal unicode escape
// C:\users\luser\Test.java
```

```
// C:\users\luser\Test.java
public class Test {
    public static void main(String [] a) {
        println ("Hello Universe!");
    }
} }
```

```
> javac Test.java
Test.java:1: illegal unicode escape
// C:\users\luser\Test.java
   ^
Test.java:6: 'class' or 'interface' expected
} }
  ^
Test.java:7: 'class' or 'interface' expected

^
Test.java:4: cannot resolve symbol
symbol  : method println  (java.lang.String)
location: class Test
        println ("Hello World");
        ^
4 errors
```

Scanning error

Parsing errors

Static semantic errors

Defining the Java Language

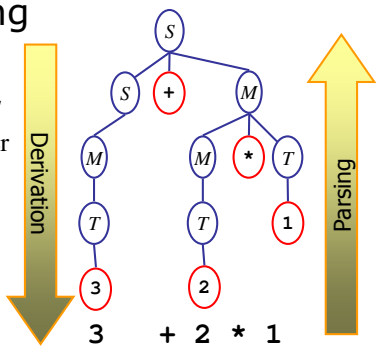
{ w | w can be generated by the CFG for Java in the Java Language Specification }

{ w | a correct Java compiler can build a parse tree for w }

Parsing

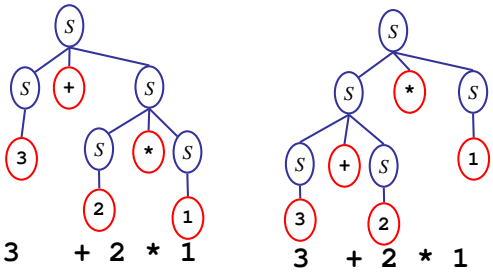
$S \rightarrow S + M \mid M$
$M \rightarrow M * T \mid T$
$T \rightarrow (S) \mid \text{number}$

Programming languages are (should be) designed to make parsing easy, efficient, and unambiguous.



Unambiguous

$S \rightarrow S + S \mid S * S \mid (S) \mid number$



Ambiguity

How can one determine if a CFG is ambiguous?

Super-duper-challenge problem: create a program that solve the "is this CFG ambiguous" problem:
Input: CFG
Output: "Yes" (ambiguous)/"No" (unambiguous)

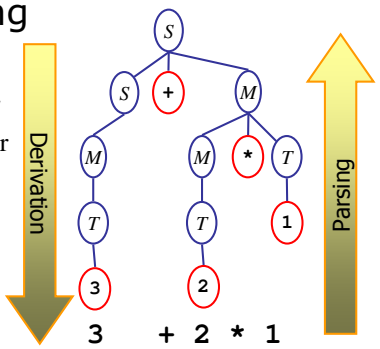
Warning: Undecidable Problem Alert!
(Not only can you not do this, it is impossible for any program to do this.)

(We will cover undecidable problems after Spring Break)

Parsing

$S \rightarrow S + M \mid M$
$M \rightarrow M * T \mid T$
$T \rightarrow (S) \mid \text{number}$

Programming languages are (should be) designed to make parsing **easy**, **efficient**, and unambiguous.



"Easy" and "Efficient"

- "Easy" - we can automate the process of building a parser from a description of a grammar
- "Efficient" – the resulting parser can build a parse tree quickly (linear time in the length of the input)

Recursive Descent Parsing

$S \rightarrow S + M \mid M$
$M \rightarrow M * T \mid T$
$T \rightarrow (S) \mid \text{number}$

```
Parse() { S(); }
S() {
  try { S(); expect("+"); M(); }
      catch { backup(); }
  try { M(); } catch {backup(); }
  error(); }
M() {
  try { M(); expect("*"); T(); } catch …
  try { T(); } catch { backup(); }
  error (); }
T() {
  try { expect("("); S(); expect(")"); } catch …;
  try { number(); } catch …; }
```

Advantages:
- Easy to produce and understand
- Can be done for any CFG

Problems:
- Inefficient (might not even finish)
- "Nondeterministic"

LL(k) (Lookahead-Left)

- A CFG is an LL(k) grammar if it can be parser deterministically with ≤ tokens lookahead

$S \rightarrow S + M \mid M$
$M \rightarrow M * T \mid T$
$T \rightarrow (S) \mid \text{number}$

LL(1) grammar

1	+	2
$S \rightarrow S + M$	$S \rightarrow S + M$	
$S \rightarrow M$		

Look-ahead Parser

```
Parse() { S(); }
S() {
  if (lookahead(1, "+")) { S(); eat("+"); M(); }
  else { M();}
M() {
  if (lookahead(1, "*")) { M(); eat("*"); T(); }
  else { T(); } }
T() {
  if (lookahead(0, "(")) { eat("("); S(); eat(")"); }
  else { number();}
```

$S \rightarrow S + M \mid M$
$M \rightarrow M * T \mid T$
$T \rightarrow (S) \mid \text{number}$

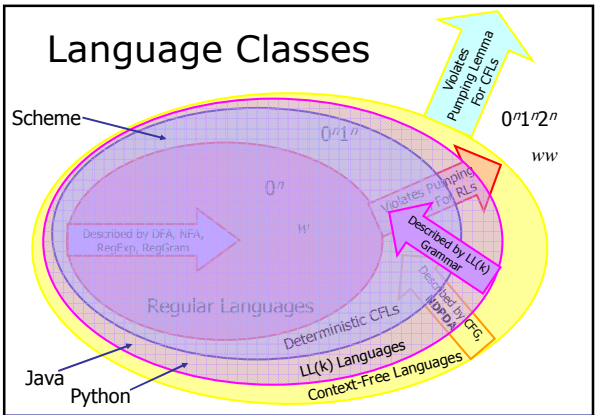
JavaCC

https://javacc.dev.java.net/

- Input: Grammar specification
- Output: A Java program that is a recursive descent parser for the specified grammar

Doesn't work for all CFGs: only for LL(k) grammars

Language Classes



Next Week

- Monday (2): Office Hours (Qi Mi in 226D)
- Monday (5:30): TA help session
- Tuesday's class (Pieter Hooimeyer): starting to get outside the yellow circle: using grammars to solve security problems
- Wednesday (9:30am): Office Hours (Qi Mi in 226D)
- Wednesday (6pm): TAs' Exam Review
- Thursday: exam in class