cs3102: Theory of Computation

Class 9:
**Context-Free Languages Contextually**


Spring 2010
University of Virginia
David Evans


# Menu

- PS2
- Recap: Computability Classes, CFL Pumping
- Closure Properties of CFLs
- Parsing

# Problem 5: PRIMES

Use the pumping lemma to prove the language,
*PRIMES* = { $1^p$ | $p$ is a prime number }
is non-regular.

Assume *PRIMES* is regular. Then, there is a DFA *M* with pumping length *p* that recognizes *PRIMES*.

***All*** RL pumping lemma proofs can start like this!

Next: pick *s*.

$s = 111$ $|s| \geq p$

$s = 1^p$ $s \in A$

$s = 1^k$

$k \geq p$ and $k$ is prime

# Problem 5: PRIMES

Use the pumping lemma to prove the language,
*PRIMES* = { $1^p$ | $p$ is a prime number }
is non-regular.

Assume *PRIMES* is regular. Then, there is a DFA *M* with pumping length *p* that recognizes *PRIMES*.

Choose $s = 1^r$ where $r$ is some prime number $\geq p$.
$s$ satisfies the requirements: $s \in$ *PRIMES* and $|s| \geq p$

Next: show for **any** choice of $xyz$ where $s = xyz$, $|xy| \leq p$ and $|y| \geq 1$, there is some $i$ where $xy^iz \notin$ *PRIMES*.

**Problem 9: Regular Grammars.** A *regular grammar* is a replacement grammar in which all rules have the form $A \rightarrow aB$ or $A \rightarrow a$ where $A$ and $B$ represent any variable and $a$ represents a terminal. Prove that all regular languages can be recognized by a regular grammar.

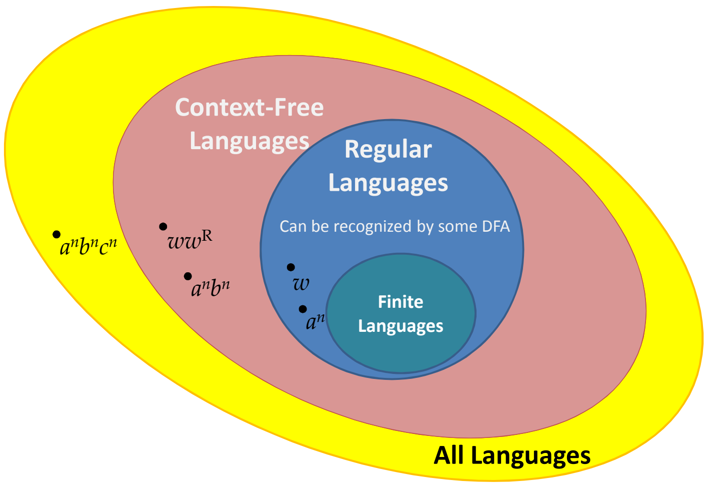
Why is this impossible?

$\emptyset$ $S \rightarrow aS$

$\{\epsilon\}$ $\Sigma = \{a, \ldots\}$

Broken definition of *regular grammar*: must also allow A → ε.

Please read the PS2 Comments thoroughly!




**Pumping Lemma for Context Free Languages:**

**Player 1:** picks $p$

**Player 2:** picks $s \in A$, $|s| \geq p$

**Player 1:** picks $u,v,x,y,z$ such that $s = uvxyz$ and $|vy| > 0$ and $|vxy| \leq p$.

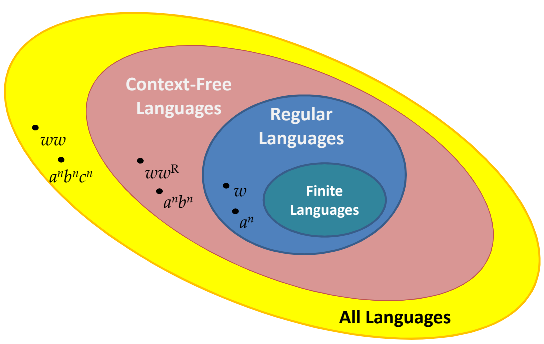
**Player 2:** picks $i \geq 0$.

Player 2 wins if $uv^ixy^iz \notin A$. If Player 2 can always win, $A$ is **not context free!**

## Example: $\{ww | w \in \Sigma^*\}$

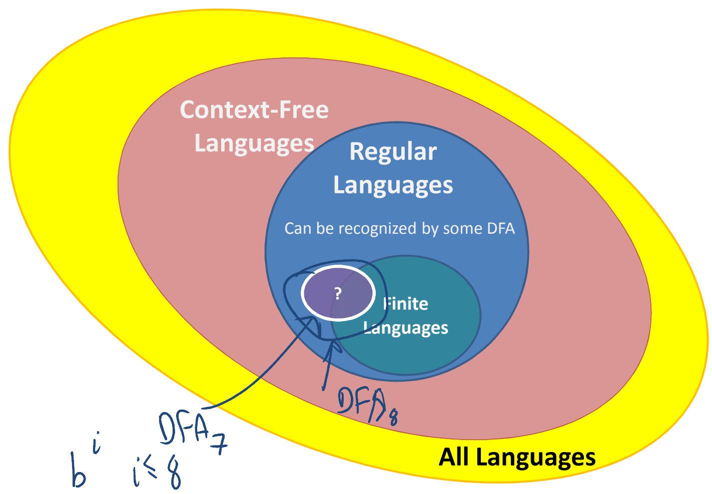
$s = \cancel{a^p a^p}\ a^p b^p a^p b^p$

$s = u\underbrace{vxy}_{\leq p}z$




## How many language classes are there?

Pirahã: *one*, *two*, *many*
Computer Sciencese: *zero*, *one*, *infinity*





## Closure Properties of RLs

If $A$ and $B$ are regular languages then:

$A^R$ is a regular language: closed under reversal
  Construct the reverse NFA

$A^*$ is a regular language
  Add a transition from accept states to start

$\overline{A}$ is a regular language (complement)
  $F' = Q - F$

$A \cup B$ is a regular language
  Construct an NFA that combines two DFAs

$A \cap B$ is a regular language
  Construct a DFA combining states from two DFAs that accepts if **both** accept

## Closure Properties of CFLs

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

$A^R$ is a context-free language ?

$A^*$ is a context-free language ?

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

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

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

Some of these are true. Some of them are false.

## CFLs Closed Under Reverse?

Given a CFL $A$, is $A^R$ a CFL?

$V \to aB \Rightarrow V \to Ba$

## CFLs Closed Under Reverse

Given a CFL $A$, is $A^R$ a CFL?

**Proof-by-construction**:
Since $A$ is a CFL, there is some CFG $G$ that recognizes $A$.
There is a CFG $G^R$ that recognizes $A^R$.

$G = (V, \Sigma, R, S)$

$G^R = (V, \Sigma, R^R, S)$

$R^R = \{ A \to \alpha^R \mid A \to \alpha \in R \}$

## CFLs Closed Under *?

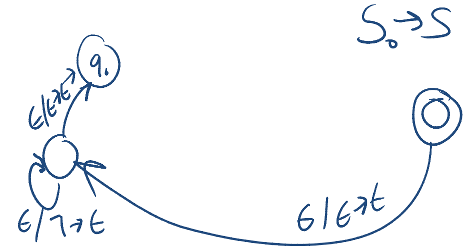
Given a CFL $A$, is $A^*$ a CFL?

$S_0 \to \varepsilon$

$S_0 \to S S_0$

$S_0 \to S$

$S \to \ldots$



aabb

ab

## CFLs Closed Under *

Given a CFL $A$, is $A^*$ a CFL?

**Proof-by-construction:** Since $A$ is a CFL, there is some CFG $G = (V, \Sigma, R, S)$ that recognizes $A$. There is a CFG $G^*$ that recognizes $A^*$:

$G^* = (V \cup \{S_0\}, \Sigma, R^*, S_0)$

$R^* = R \cup \{ S_0 \to S \} \cup \{ S_0 \to S_0 S_0 \} \cup \{ S_0 \to \varepsilon \}$

## 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

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

Is $A \cup B$ is a context-free language ?

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

Is $AB$ is a context-free language?

**Left for you on PS3.**

## CFLs Closed Under Union

Given two CFLs $A$ and $B$ is $A \cup B$ a CFL?

## CFLs Closed Under Union

**Proof-by-construction:** There is a CFG $G_{AUB}$ that recognizes $A \cup B$. Since $A$ and $B$ are CFLs, there are CFGs $G_A = (V_A, \Sigma_A, R_A, S_A)$ and $G_B = (V_B, \Sigma_B, R_B, S_B)$ that generate $A$ and $B$.

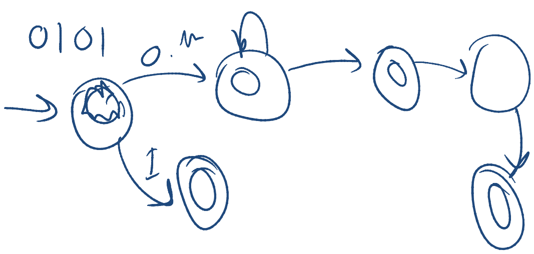
$$G_{AUB} = (V_A \cup V_B, \Sigma_A \cup \Sigma_B, R_{AUB}, S_0)$$
$$R_{AUB} = R_A \cup R_B \cup \{\ S_0 \rightarrow S_A\ \} \cup \{\ S_0 \rightarrow S_B\ \}$$

(Assumes $V_A$ and $V_B$ are disjoint which is easy to arrange by changing variable names.)

## CFLs Closed Under Complement?

$\{\mathbf{0}^i\mathbf{1}^i \mid i \geq 0\ \}$ is a CFL. $\Sigma = \{0,1\}$

Is its complement?




## CFLs Closed Under Complement?

$\{\mathbf{0}^i\mathbf{1}^i \mid i \geq 0\ \}$ is a CFL. $\Sigma = \{0,1\}$

Is its complement?

**Yes.** We can make a DPDA that recognizes it: swap accepting states of DPDA that recognizes $\mathbf{0}^i\mathbf{1}^i$.

Not a counterexample…but not a proof either.

## Complementing Non-CFLs

$\{ww \mid w \in \Sigma^*\ \}$ is **not** a CFL.

Is its complement?

$S \rightarrow S_{odd} \cup S_{even}$ $\quad \Sigma = \{0,1\}$

√

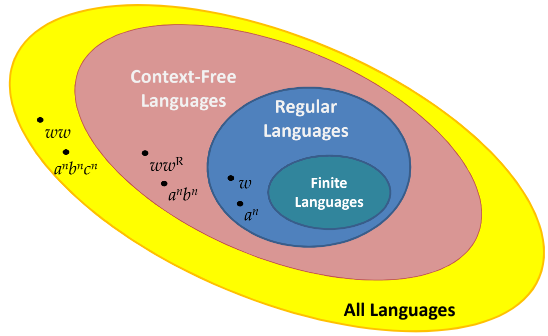
## CFG for $\overline{L_{ww}}$ $(L_{\neg ww})$

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

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

$S_{\text{Odd}} \rightarrow PS_{\text{Odd}} \mid \mathbf{0} \mid \mathbf{1}$
$P \rightarrow \mathbf{00} \mid \mathbf{01} \mid \mathbf{10} \mid \mathbf{11}$

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


## Where is Java?

## What is the Java Programming Language?

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

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

$s \in JAVA$

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

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

$s \notin JAVA$

## Defining the Java Language

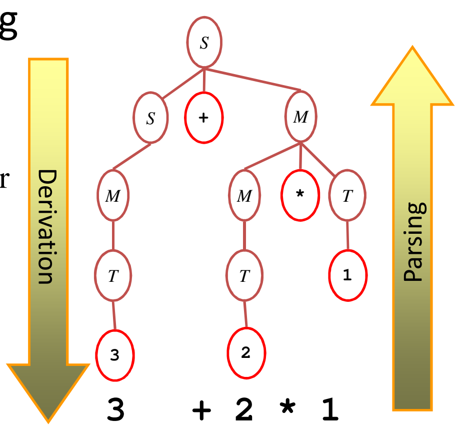
$JAVA$ = { $w$ | $w$ can be generated by the CFG for Java in the Java Language Specification }

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

## Parsing

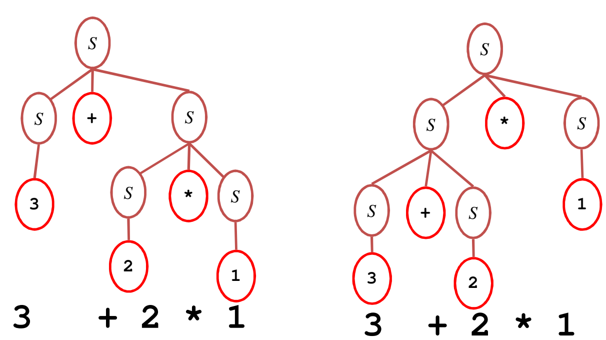
$S \rightarrow S + M \mid M$
$M \rightarrow M * T \mid T$
$T \rightarrow (S) \mid \text{number}$

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 (automatic A++):** create a program that solve the "is this CFG ambiguous" problem:

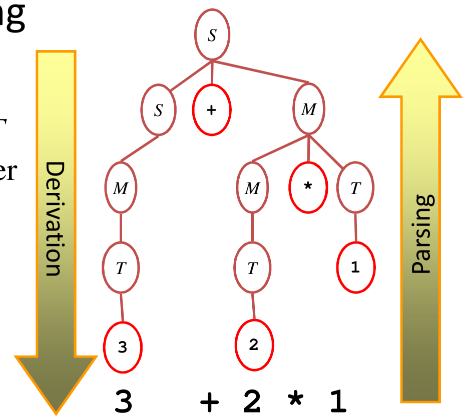
**Input:** any CFG
**Output:** "Yes" (ambiguous)/"No" (unambiguous)

**Warning: Undecidable Problem Alert!**
Don't slack off on the rest of the course thinking you can solve this. It is **known to be impossible!**

# 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

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

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

**Easy** to produce and understand
**Works for any CFG**
**Inefficient** (might not even finish)

# LL($k$) (Lookahead-Left)

A CFG is an LL($k$) grammar if it can be parser deterministically with $\leq k$ tokens lookahead

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

| 1 | + | 2 |
|---|---|---|
| $S \rightarrow S + M$<br>$S \rightarrow M$ | $S \rightarrow S + M$ | |

LL(1) grammar

# 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();}
```

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

**Fairly easy** to produce automatically
**Efficient (for low lookahead)**
**Doesn't work for all CFGs**

# JavaCC

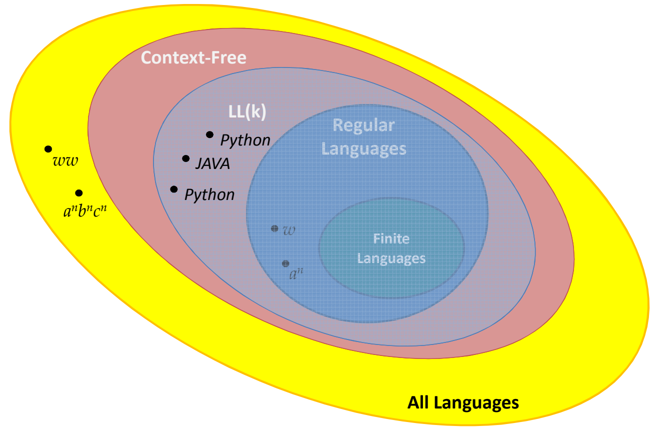
https://javacc.dev.java.net/

**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

Similar tools exist for all major programming languages:
Lex/Flex + YACC/Bison (C): "Yet another compiler compiler"
PLY (Python): Python lex/yacc
ANTLR




# Language Classes

# Return PS2

front of room

jth2ey (James Harrison) - pmc8p

ras3kd (Robyn Short) – yyz5w

afg2s (Arthur Gordon) – dk8p

dr7jx (David Renardy) – jmd9xk

# Charge

- **Read PS2 Comments**
- **PS3 due Tuesday**