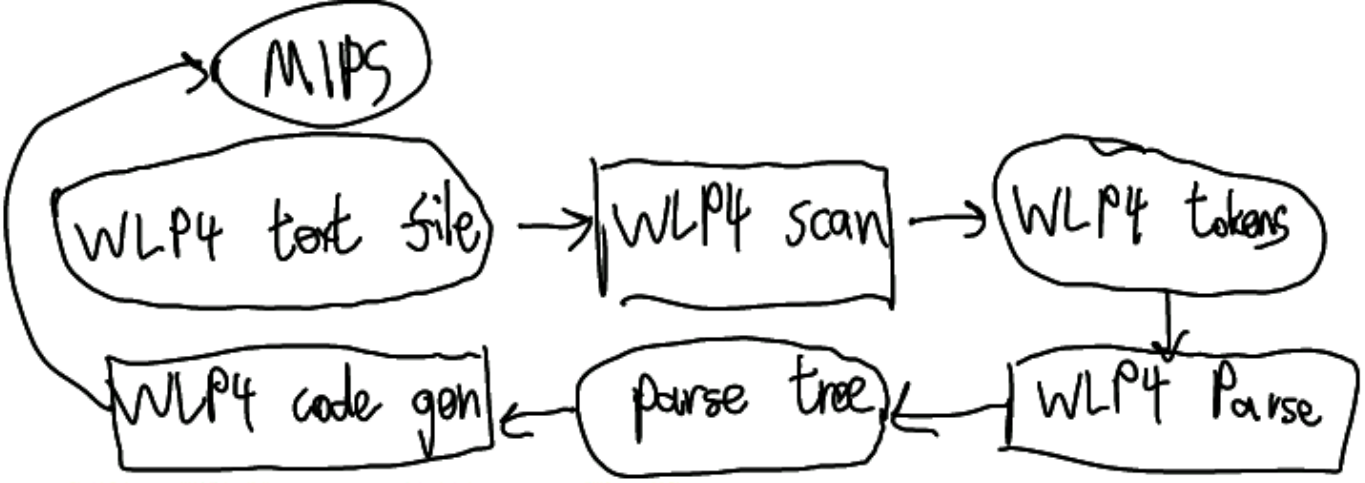
Lecture 13+14

Context-Free Languages

CS 241: Foundations of Sequential Programs
Fall 2014

Troy Vasiga et al
University of Waterloo



Big Picture of Compilation

Big picture:

- lexical analysis → scanning + tokenizing
- syntactic analysis
  ↳ parsing
- context-sensitive (semantic) analysis
  ↳ meaning
- synthesis (code geneneration)

Other points:

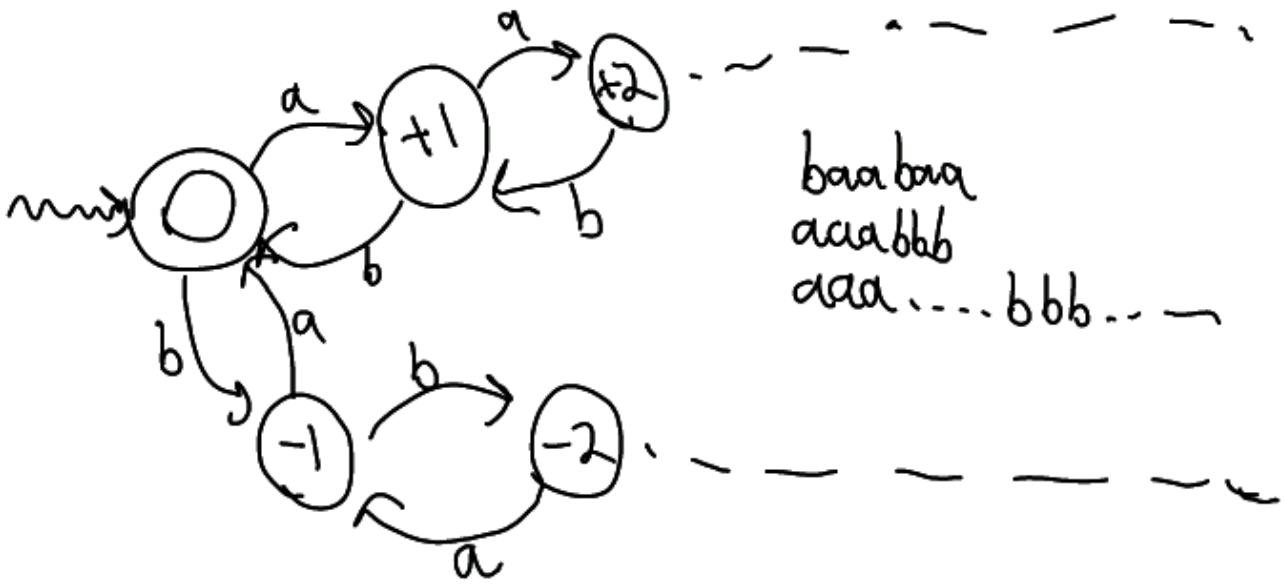
- staging can improve error messages
- staging uses the "right tool for the job"
- doing extra-work at an early stage is possible but over-complicating

Non-regular languages

Give a DFA over $\Sigma = \{a, b\}$ for
↳ finite

L = {w: numbers of a's in w = number of b's in w}

ε ∈ L abba ∈ L aba ∉ L



N states

aa...a bb...b
N+3 N+3

Proof: CS360

Context-Free Languages

- L is not a regular language, but it is *context-free*
- Note that a compiler needs to do this and then some:

```
int foo(int a, int b)
if (( )(( ))
   aa b aa bbb
```

- Context-free languages are built from:
  - finite sets
  - concatenation
  - union
  - recursion

same as reg. lang.

not the same as reg. lang.

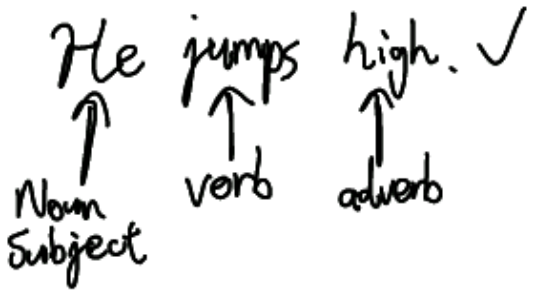
- recognizers for regular languages use a finite amount of memory
- recognizers for context-free languages use a finite amount of memory plus one (unbounded) stack

scanner → DFA

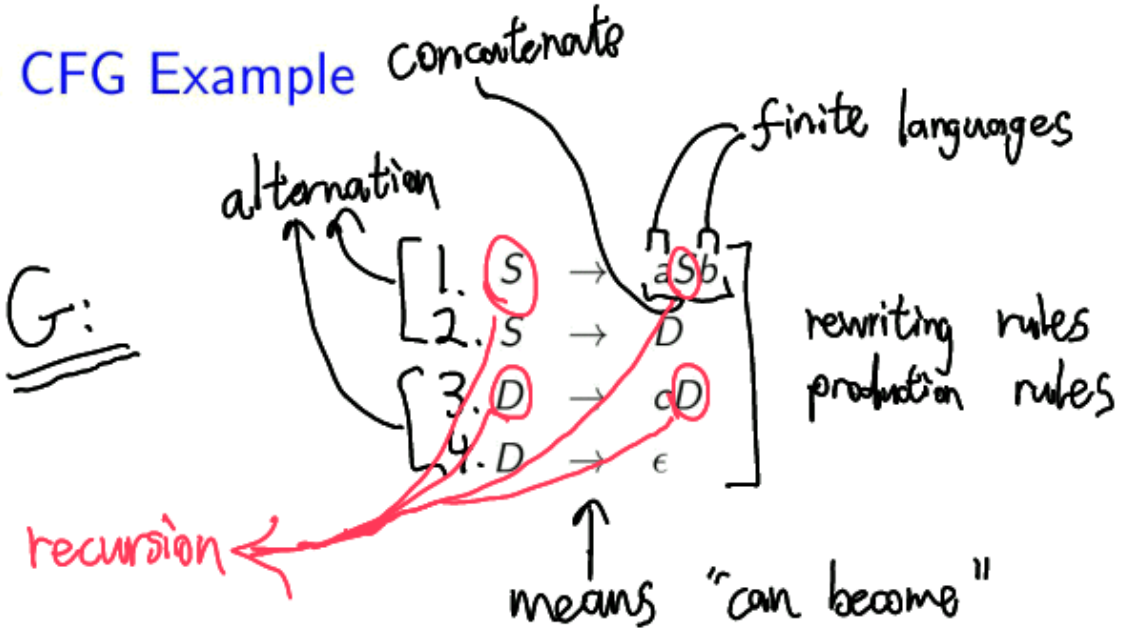
parser → DFA + STACK + current state

Context-Free Grammars

A way to specify a context-free language.



A CFG Example



---

word $accb \in L(G)$

Derivation:

$$S \overset{①}{\Rightarrow} aSb \overset{②}{\Rightarrow} aDb \overset{③}{\Rightarrow} acDb \overset{③}{\Rightarrow} accDB \overset{④}{\Rightarrow} accB$$

Discussion of example

- G: a context free grammar
- L(G): the language specified by CFG G
  - language → set of words
- a word:
  a sequence of tokens that can be derived by CFG G
- a derivation:
  (informally) a sequence of rewriting steps from G
- alternation and concatenation
- recursion vs. repetition
  - recursion → is strictly more powerful than repetition

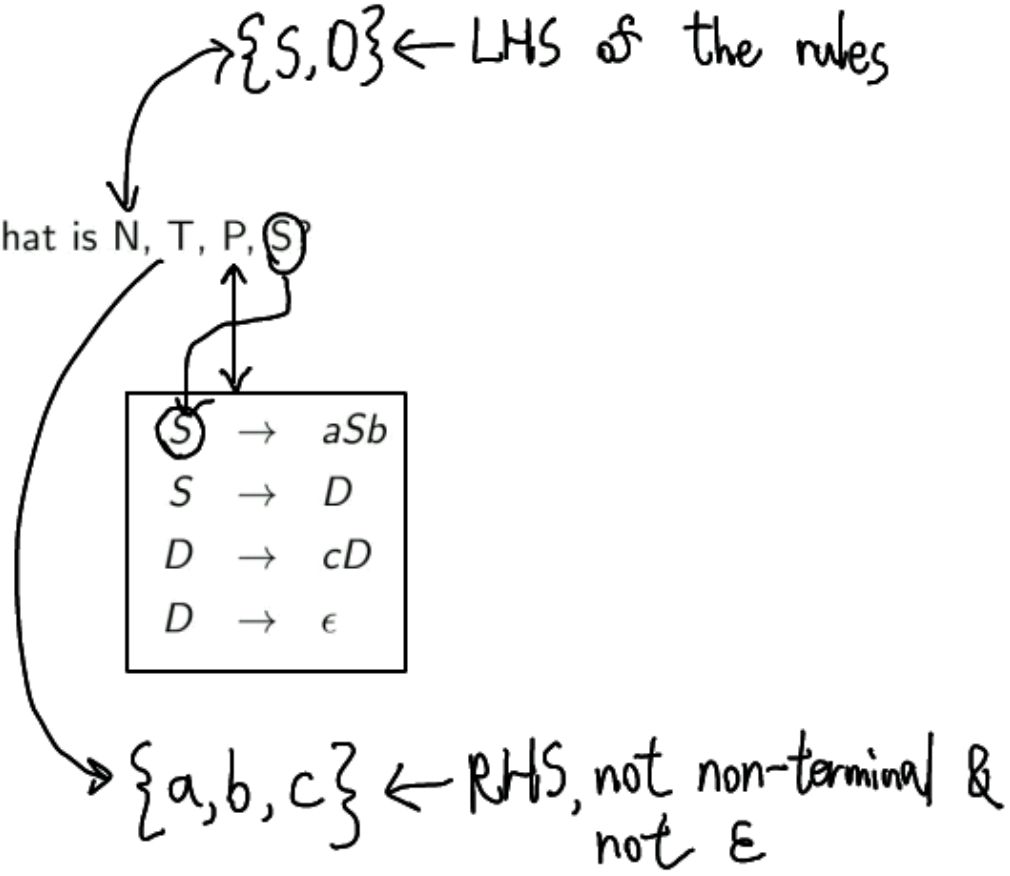
Formal Definition

A context-free grammar (CFG) consists of

- N – a finite set of non-terminals
  - "not the ending"
- T – a finite set of terminals
  - "ending"
- P – a finite set of production rules
  $A \to B$, $A \in N$, $B \in (N \cup T)^*$
- S – start symbol, $S \in N$
  - by convention: S is the LHS of the first rule

Example, more formally

From the example, what is N, T, P, S?



Example: balanced parentheses

Example words:

ε, (), (( )), ()(), ...

CFG:

1. $S \to (S)$
2. $S \to \varepsilon$
3. $S \to SS$

A sample derivation: (()())

$S \overset{①}{\Rightarrow} (S) \overset{③}{\Rightarrow} (SS) \overset{①}{\Rightarrow} ((S)S) \overset{①}{\Rightarrow} ((S)(S)) \overset{②}{\Rightarrow} ((\,)(S)) \overset{②}{\Rightarrow} ((\,)(\,))$

Binary expressions

Words are composed of binary numbers (no leading zeros, other than 0) with + or - signs in infix notation.

Example words: 1001, 10+1, 11-11110+0

CFG:

1. $E \to E + E$
2. $E \to E - E$
3. $E \to B$
4. $B \to \emptyset$
5. $B \to D$
6. $D \to 1$
7. $D \to D\emptyset$
8. $D \to D1$

E = expression
B = binary #
D = digits (non-zero)

Derivations

Two derivations of 10+1 using the previous grammar:

Leftmost derivation

$E \overset{1}{\Rightarrow} E+E$

$\overset{3}{\Rightarrow} B+E$

$\Rightarrow D+E$

$\Rightarrow D\emptyset+E$

$\Rightarrow 1\emptyset+E$

$\Rightarrow 1\emptyset+B$

$\Rightarrow 1\emptyset+D$

$\Rightarrow 1\emptyset+1$

Rightmost derivation

$E \overset{1}{\Rightarrow} E+E$

$\overset{3}{\Rightarrow} E+B$

$\Rightarrow E+D$

$\Rightarrow E+1$

$\Rightarrow B+1$

$\Rightarrow D+1$

$\Rightarrow D\emptyset+1$

$\Rightarrow 1\emptyset+1$

Formal Definitions

We say that $\alpha A \beta$ *directly derives* $\alpha\gamma\beta$ if there exists a production $A \to \gamma$.
Also called a *derivation step*.

$A \in N$

$\alpha, \beta, \gamma \in (N \cup T)^*$

$\alpha A \beta \Rightarrow \alpha \gamma \beta$

We say that $\alpha A \beta$ *derives* $\alpha\gamma\beta$ if

$\alpha A \beta \Rightarrow \alpha\theta_1\beta \Rightarrow \alpha\theta_2\beta \Rightarrow \ldots \Rightarrow \alpha\gamma\beta$

write this as $\alpha A \beta \Rightarrow^* \alpha\gamma\beta$

Formal Defintions (continued)

G derives $w \in T^*$ if $S \Rightarrow^* w$

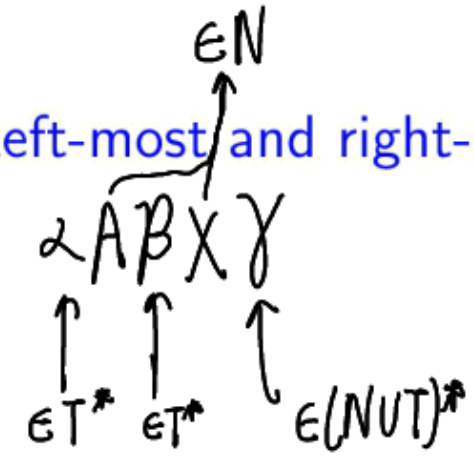
$L(G) = \{w \in T^* : S \Rightarrow^* w\}$

↑ language specified by G

L is context-free if

$\exists$ a CFG $\cdot$ $L(G) = L$

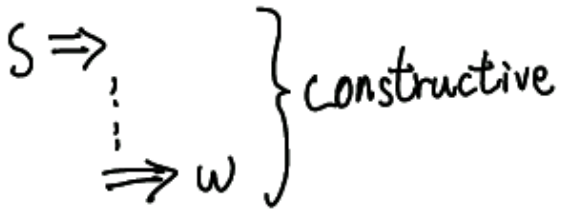
Left-most and right-most derivations (formally)



a leftmost derivation must rewrite A

Derivations as Proofs

Given a CFG G & word w



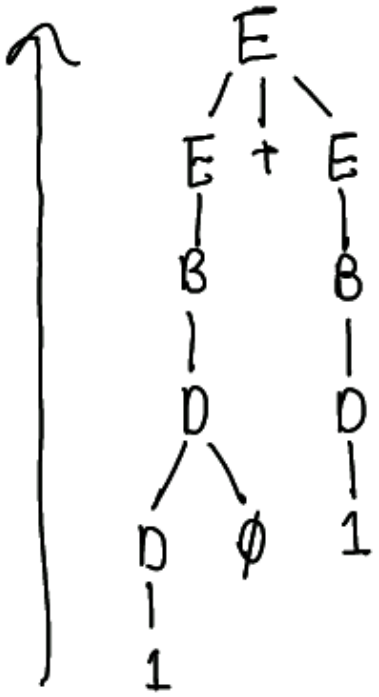
Parse Trees

Example:

10+1

using

G_B



Internal nodes:
Non-terminal

Leaf nodes:
Terminals

Discussion:

All derivatives of 10+1 are in this parse tree

Meaning of a parse tree

"Recursive Descent Parsing"

"Syntax Directed Translation"

$E \to B \iff E.val = B.val$

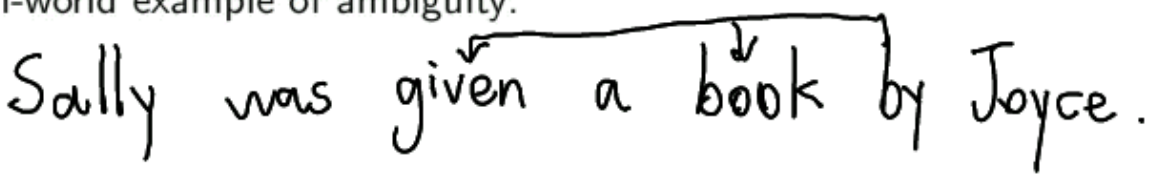
$D_1 \to D_2 \emptyset \iff D_1.val = D_2.val * 2$

$D \to 1 \iff D.val = 1$

$10+1$ means 3

Problems that grammars can encounter: Ambiguity

A real-world example of ambiguity:



Ambiguity in CFGs: Formal Definition

- A string x is ambiguous if...

  there is more than one parse tree for x.

- A CFG G is ambiguous if...

  there is at least one string $x \in L(G)$ which is ambiguous.

Problems that grammars can encounter: Ambiguity

Consider the binary expression grammar, and 1-10+11

meaning: 2

"1-2+3" ⇒ "2"

meaning: -4

Problems that grammars can encounter: Ambiguity

Terminology:

also may have:
≥ 2 leftmost derivations
≥ 2 rightmost derivations
for the same word

Left-recursion
+
Right-recursion
⇓
ambiguous

Rightmost symbol of RHS is the LHS

Right recursion

Fixing ambiguity

Let's rewrite our binary expression grammar differently.

$E \to E + E$

1. $E \to B + E$
2. $E \to B - E$
3. $E \to B$
4. $E \to 0$
5. $B \to D$
6. $D \to 1$
7. $D \to DD$
8. $D \to D1$

Left recursion

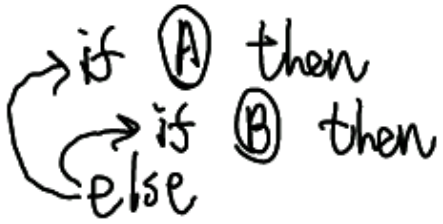
Leftmost symbol of RHS is the LHS

Same as before

unambiguous

Ambiguity in programming languages

- Some programming languages have ambiguous grammars
- Pascal has a "dangling else"



- "Fixed" by way of a footnote

RTFM

Associativity

right associative

1-10+11

the parse tree

means: -4

only one parse tree :)

...it's the wrong tree :(

Fixing associativity problems

1. $E \to E + B$ ← left-recursion ⇔ left-associative
2. $E \to E - B$
3. same as before
4.
5.
6.
7.
8.

means: 2 :)

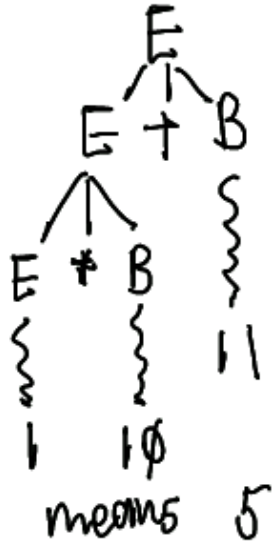
Precedence Problems

Adding multiplication to the grammar.

0. E → E * B

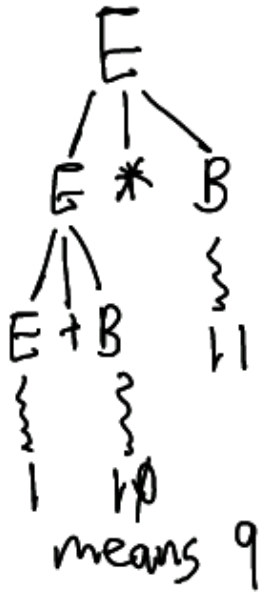
Consider 1*10+11.

means: 5



Consider 1+10*11

means 7



Fixing precedence problems: grammar

nearer the root ⇒ later the evaluation
⇒ lower precedence

$S \to S + P$

$S \to S - P$

$S \to P$

$P \to P * B$

$P \to P / B$

$P \to B$

$B \to \ldots$ (as before)

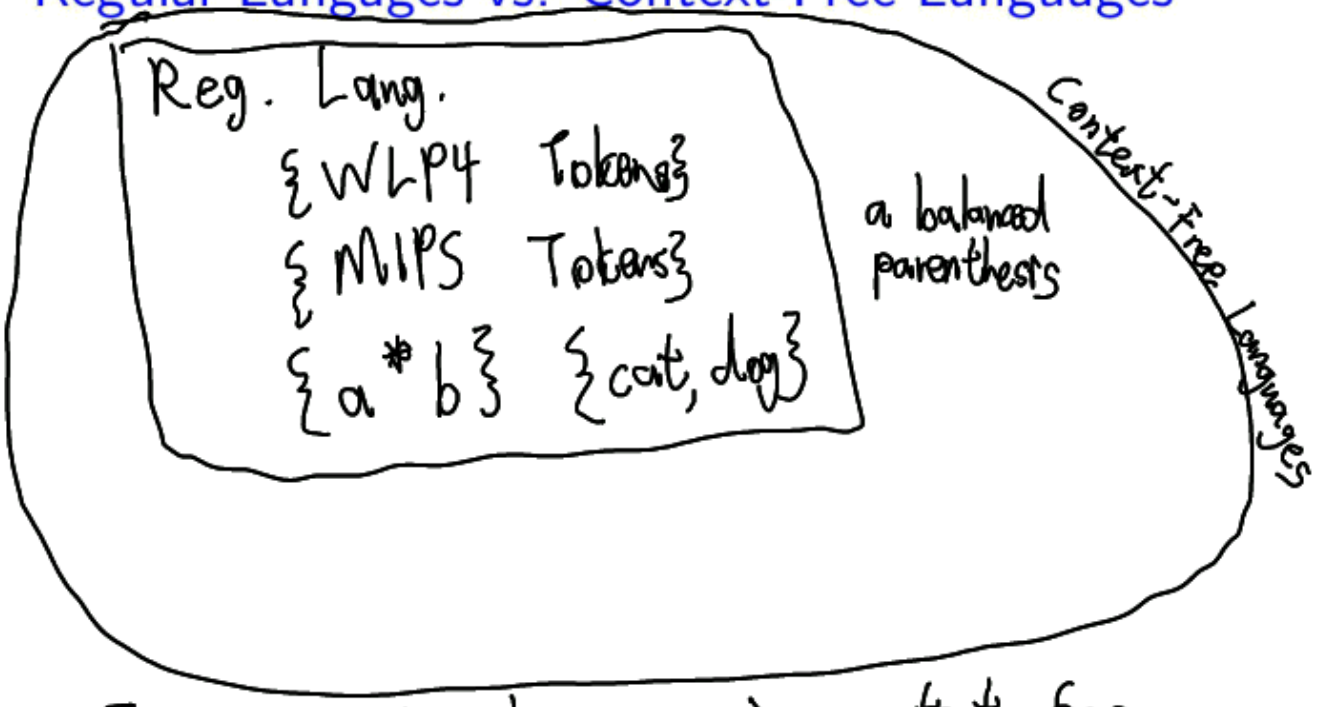
S = sum
P = product

Fixing precedence problems: parse tree

11 + 100 * 110 + 10 ⇒ means: 29

Ex. add ( )
to the grammar
(10+11)*10

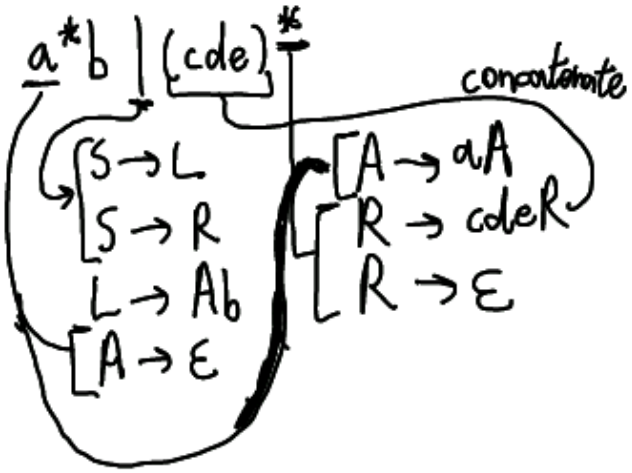
Regular Langages vs. Context-Free Languages



Every regular language is context-free.

"Proof": ∃ regular expression for the reg. lang.

e.g. $a^*b|(cde)^*$ ⇐ write a CFG for this



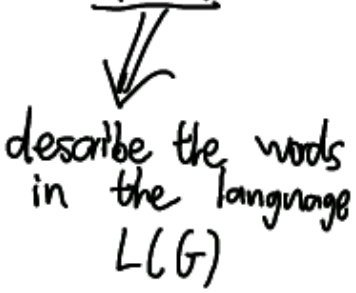
∴ All reg. lang. are context free.

Assignment Tips

- WLP4 grammar ← read it
- `.cfg` file format ← read the spec
- recursion is your friend
  - → CFG has recursive rule

Summary

- CFG are a way to specify a language



- To recognize ⇒ would like a derivation
  ([Is $w \in L(G)$? to answer, I need $S \Rightarrow^* w$
  like to do this automatically