



# Theory of Computation

## Week 6

**Much of the material on this slides comes from the recommended textbook by Elaine Rich**

# Announcements

## ❑ Midterm

❑ **Schedule:** 06/04/2020 (Tuesday) 8:00 ~ 9:00 (online)

❑ **Syllabus:** Topics covered in Week 01 to Week 05 (lecture/tutorial)

# Detailed content

## Weekly program

- ✓ Week 1 – Background knowledge revision: logic, sets, proof techniques
- ✓ Week 2 – Languages and strings. Hierarchies. Computation. Closure properties
- ✓ Week 3 – Finite State Machines: non-determinism vs. determinism
- ✓ Week 4 – Regular languages: expressions and grammars
- ✓ Week 5 – Non regular languages: pumping lemma. Closure

### Week 6 – Context-free languages: grammars and parse trees

- ☐ Week 7 – Pushdown automata
- ☐ Week 8 – Non context-free languages: pumping lemma and decidability. Closure
- ☐ Week 9 – Decidable languages: Turing Machines
- ☐ Week 10 – Church-Turing thesis and the unsolvability of the Halting Problem
- ☐ Week 11 – Decidable, semi-decidable and undecidable languages (and proofs)
- ☐ Week 12 – Revision of the hierarchy. Safety-critical systems
- ☐ Week 13 – Extra revision (if needed)

# Week 06 Videos

## You already know

- ❑ Context Free Grammar
  - ❑ How it is different from Regular Grammar
  - ❑ Why it is called context free
  - ❑ Formal definition and example
  - ❑ Regular languages are proper subset of context free languages
- ❑ Parse Tree
  - ❑ Derivation
  - ❑ Weak/Strong generative capacity



Videos to watch before lecture



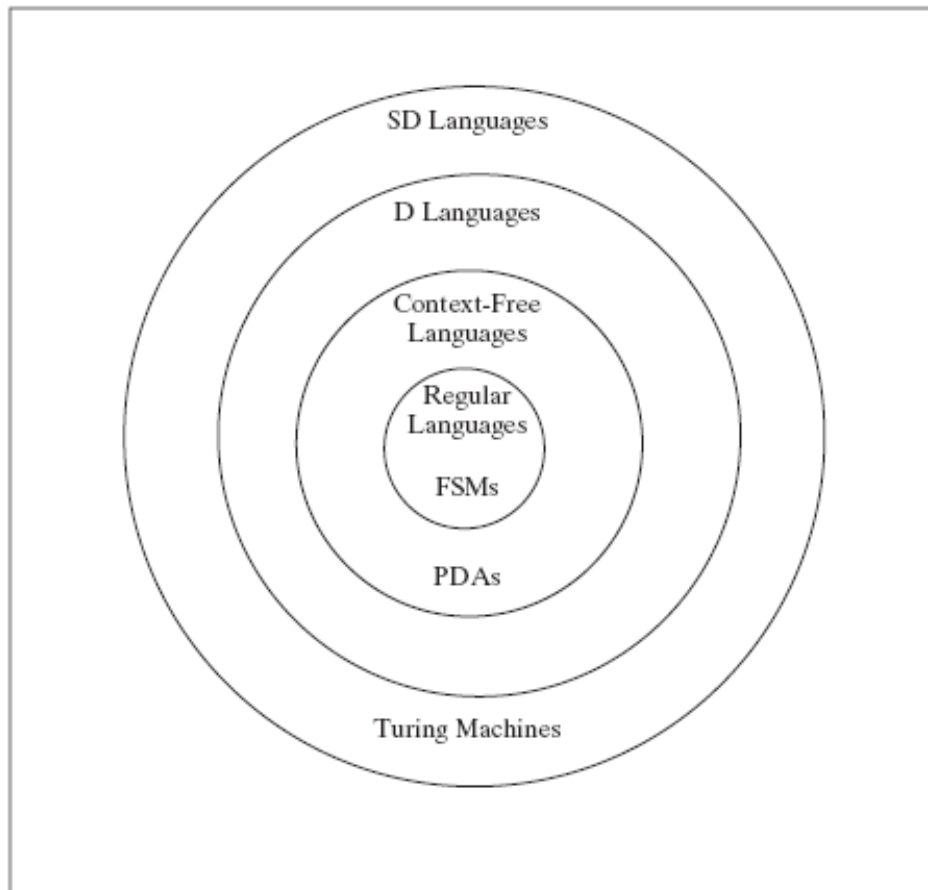
Additional videos to watch for this week

# Week 06 Lecture Outline

## Context-free languages: grammars and parse trees

- ❑ Rewrite Systems
- ❑ Grammars
  - ❑ Context Free Grammars
- ❑ Context Free Languages
- ❑ Properties of CFG
  - ❑ Recursion
  - ❑ Self-Embedding
- ❑ Backus Naur Form (BNF)
- ❑ Designing CFGs
- ❑ Simplifying CFGs
- ❑ Parse Tree
- ❑ Ambiguity

# THE HIERARCHY



# REWRITE SYSTEMS

7

A ***rewrite system*** (or ***production system*** or ***rule-based system***) is:

- a list of rules, and
- an algorithm for applying them.

Each rule has a left-hand side and a right hand side.

Example rules:

$$S \rightarrow aSb$$

$$aS \rightarrow \varepsilon$$

$$aSb \rightarrow bSabSa$$

# SIMPLE REWRITE

8

$simple\text{-}rewrite(R: \text{rewrite system}, w: \text{initial string}) =$

1. Set *working-string* to  $w$ .
2. Until told by  $R$  to halt do:
  - a. Match the LHS of some rule against some part of *working-string*.
  - b. Replace the matched part of *working-string* with the RHS of the rule that was matched.
3. Return *working-string*.

If  $simple\text{-}rewrite(R, w)$  can return some string  $s$  then we will say that  $R$  can **derive**  $s$  from  $w$  or there exists a **derivation** in  $R$  of  $s$  from  $w$ .



# SIMPLE REWRITE

A rewrite system formalism specifies:

- The form of the rules
- How *simple-rewrite* works:
  - How to choose rules?
  - When to quit?

# SIMPLE REWRITE

10

$$w = SaS$$

Rules:

- [1]  $S \rightarrow aSa$
- [2]  $S \rightarrow bSb$
- [3]  $aS \rightarrow \varepsilon$

What order to apply the rules?

When to quit?

# RULE BASED SYSTEMS

11

- Expert systems
- Cognitive modeling
- Business practice modeling
- General models of computation
- **Grammars**



[Source: Wikipedia]

# GRAMMARS



13

A grammar is a set of rules that are stated in terms of two alphabets:

- a ***terminal alphabet***,  $\Sigma$ , that contains the symbols that make up the strings in  $L(G)$ , and
- a ***nonterminal alphabet***, the elements of which will function as working symbols that will be used while the grammar is operating. These symbols will disappear by the time the grammar finishes its job and generates a string.

A grammar has a unique **start symbol**, often called  $S$ .

# USING A GRAMMAR TO DERIVE A STRING

14

*Simple-rewrite*  $(G, S)$  will generate the strings in  $L(G)$ .

We will use the symbol  $\Rightarrow$  to indicate steps in a derivation.

A derivation could begin with:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow \dots$$

# GENERATING MANY STRINGS

15

Multiple rules may match.

Given:  $S \rightarrow aSb$ ,  $S \rightarrow bSa$ , and  $S \rightarrow \varepsilon$

Derivation so far:  $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow$

Three choices at the next step:

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabSabb$

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

(using rule 1),

(using rule 2),

(using rule 3).

# GENERATING MANY STRINGS

16

- One rule may match in more than one way.

Given:  $S \rightarrow aTTb$ ,  $T \rightarrow bTa$ , and  $T \rightarrow \varepsilon$

Derivation so far:  $S \Rightarrow aTTb \Rightarrow$

Two choices at the next step:

$S \Rightarrow a\underline{TT}b \Rightarrow abTaTb \Rightarrow$

$S \Rightarrow aT\underline{T}b \Rightarrow aTbTab \Rightarrow$



# GENERATING MANY STRINGS

17

May stop when:

1. The working string no longer contains any nonterminal symbols (including, when it is  $\varepsilon$ ).

In this case, we say that the working string is ***generated*** by the grammar.

Example:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

# GENERATING MANY STRINGS

18

May stop when:

2. There are nonterminal symbols in the working string but none of them appears on the left-hand side of any rule in the grammar.

In this case, we have a blocked or non-terminated derivation but no generated string.

Example:

Rules:  $S \rightarrow aSb$ ,  $S \rightarrow bTa$ , and  $S \rightarrow \varepsilon$

Derivations:  $S \Rightarrow aSb \Rightarrow abTab \Rightarrow$

[blocked]

# GENERATING MANY STRINGS

19

It is possible that neither (1) nor (2) is achieved.

Example:

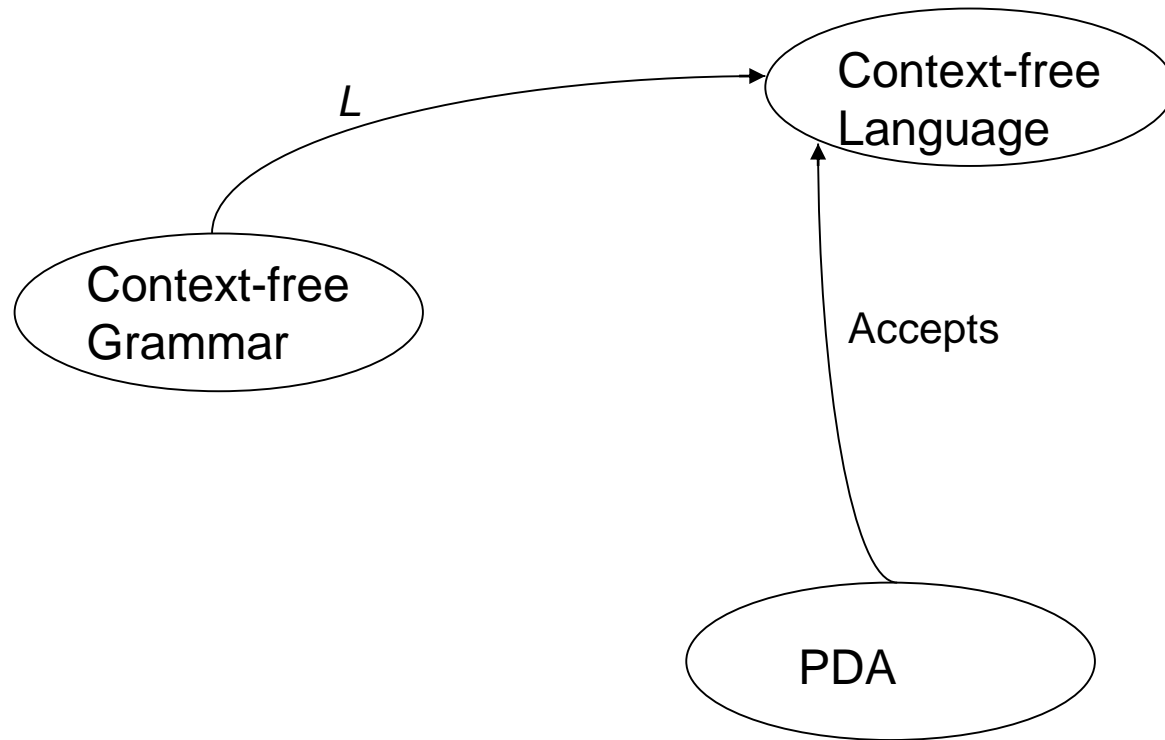
$G$  contains only the rules  $S \rightarrow Ba$  and  $B \rightarrow bB$ , with  $S$  the start symbol.

Then all derivations proceed as:

$$S \Rightarrow Ba \Rightarrow bBa \Rightarrow bbBa \Rightarrow bbbBa \Rightarrow bbbbBa \Rightarrow \dots$$

# CONTEXT-FREE GRAMMARS, LANGUAGES, AND PDAS

20



# MORE POWERFUL GRAMMARS

21

Regular grammars must always produce strings one character at a time, moving left to right.

But it may be more natural to describe generation more flexibly.

Example 1:  $L = ab^*a$

$$S \rightarrow aBa$$

$$B \rightarrow \varepsilon$$

$$B \rightarrow bB$$

vs.

$$S \rightarrow aB$$

$$B \rightarrow a$$

$$B \rightarrow bB$$

Example 2:  $L = \{a^n b^* a^n, n \geq 0\}$

$$S \rightarrow B$$

$$S \rightarrow aSa$$

$$B \rightarrow \varepsilon$$

$$B \rightarrow bB$$

Key distinction: Example 1 is not self-embedding.

# CONTEXT-FREE GRAMMARS



22

No restrictions on the form of the right hand sides.

$$S \rightarrow abDeFGab$$

But require single non-terminal on left hand side.

$$S \rightarrow$$

but not  $ASB \rightarrow$

# $A^n B^n$

**$A^nB^n$**

$$S \rightarrow \varepsilon$$

$$S \rightarrow aSb$$



# BALANCED PARENTHESES

25

March 30, 2020

**COMP2270 - Semester 1 - 2020** | [www.newcastle.edu.au](http://www.newcastle.edu.au)

# BALANCED PARENTHESES

26

$$S \rightarrow \varepsilon$$

$$S \rightarrow SS$$

$$S \rightarrow (S)$$



# CONTEXT-FREE GRAMMARS

A context-free grammar  $G$  is a quadruple,  $(V, \Sigma, R, S)$ , where:

- $V$  is the rule alphabet, which contains nonterminals and terminals.
- $\Sigma$  (the set of terminals) is a subset of  $V$ ,
- $R$  (the set of rules) is a finite subset of  $(V - \Sigma) \times V^*$ ,
- $S$  (the start symbol) is an element of  $V - \Sigma$ .

Example:

$(\{S, a, b\}, \{a, b\}, \{S \rightarrow a S b, S \rightarrow \varepsilon\}, S)$

# DERIVATIONS

28

$$\begin{array}{ccc} x \Rightarrow_G y \text{ iff } x = \alpha A \beta & \downarrow & \text{and } A \rightarrow \gamma \text{ is in } R \\ & & y = \alpha \gamma \beta \end{array}$$

$w_0 \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \dots \Rightarrow_G w_n$  is a derivation in  $G$ .

Let  $\Rightarrow_G^*$  be the reflexive, transitive closure of  $\Rightarrow_G$ .

Then the language generated by  $G$ , denoted  $L(G)$ , is:

$$\{w \in \Sigma^* : S \Rightarrow_G^* w\}.$$

# DERIVATIONS

Example:

Let  $G = (\{S, a, b\}, \{a, b\}, \{S \rightarrow a S b, S \rightarrow \varepsilon\}, S)$

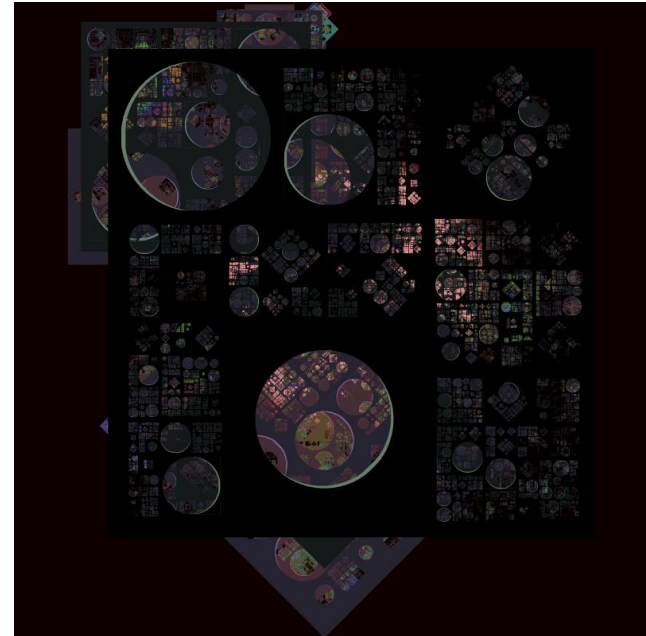
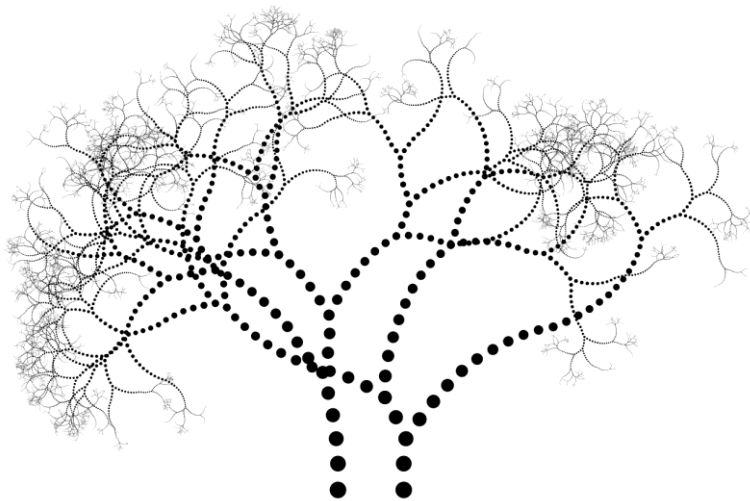
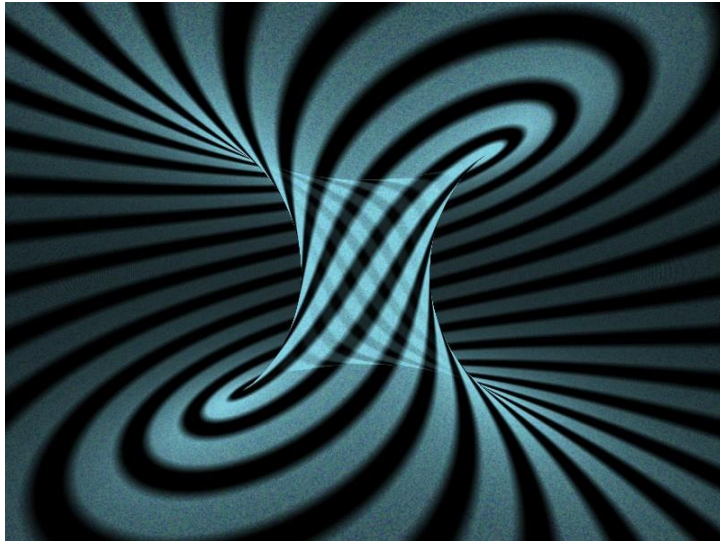
$S \Rightarrow a S b \Rightarrow aa S bb \Rightarrow aaa S bbb \Rightarrow aaabbb$

$S \Rightarrow^* aaabbb$

A language  $L$  is **context-free** if and only if it is generated by some context-free grammar  $G$ .

# Context Free Art: CFDG

31



Context Free

<http://www.contextfreeart.org/>

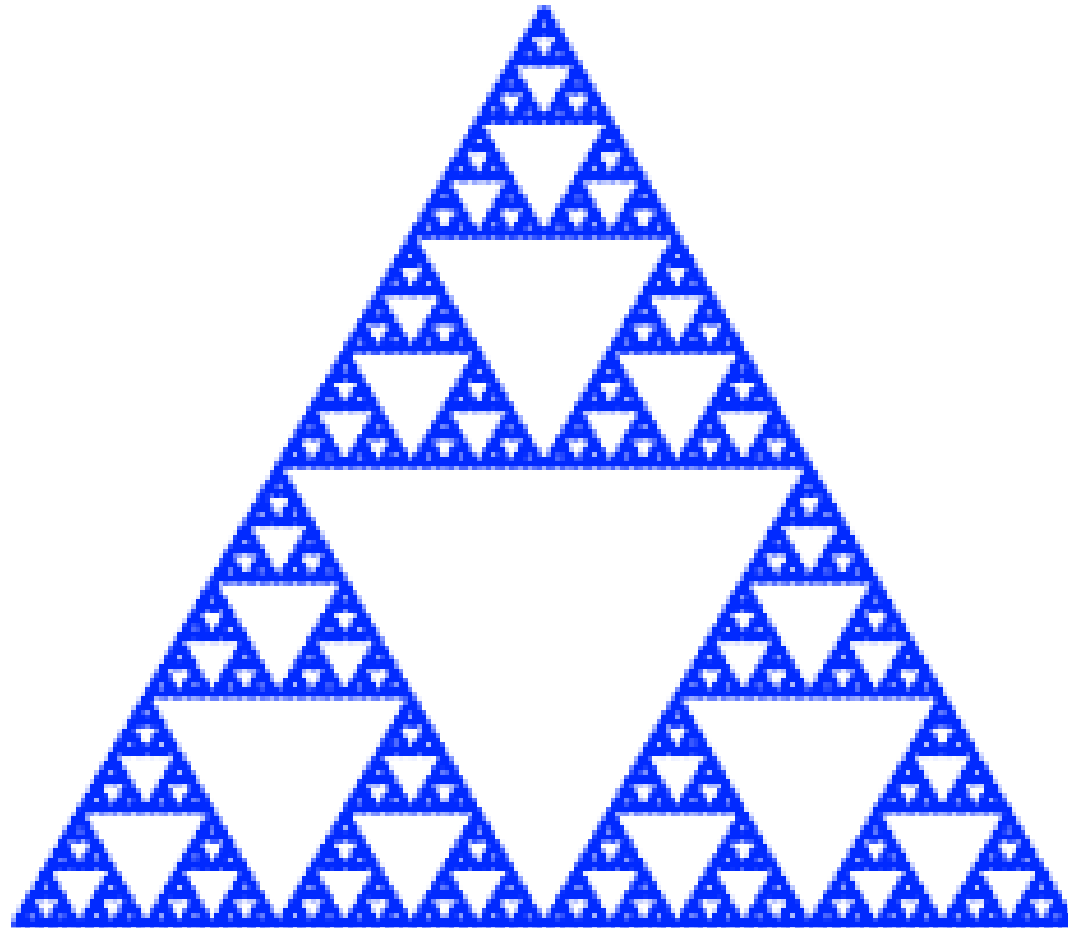
Source: <http://www.contextfreeart.org/>

March 30, 2020

COMP2270 - Semester 1 - 2020 | [www.newcastle.edu.au](http://www.newcastle.edu.au)

# Fractals:

32



March 30, 2020

COMP2270 - Semester 1 - 2020 | [www.newcastle.edu.au](http://www.newcastle.edu.au)



# Lindenmayer Systems

33

L-system:

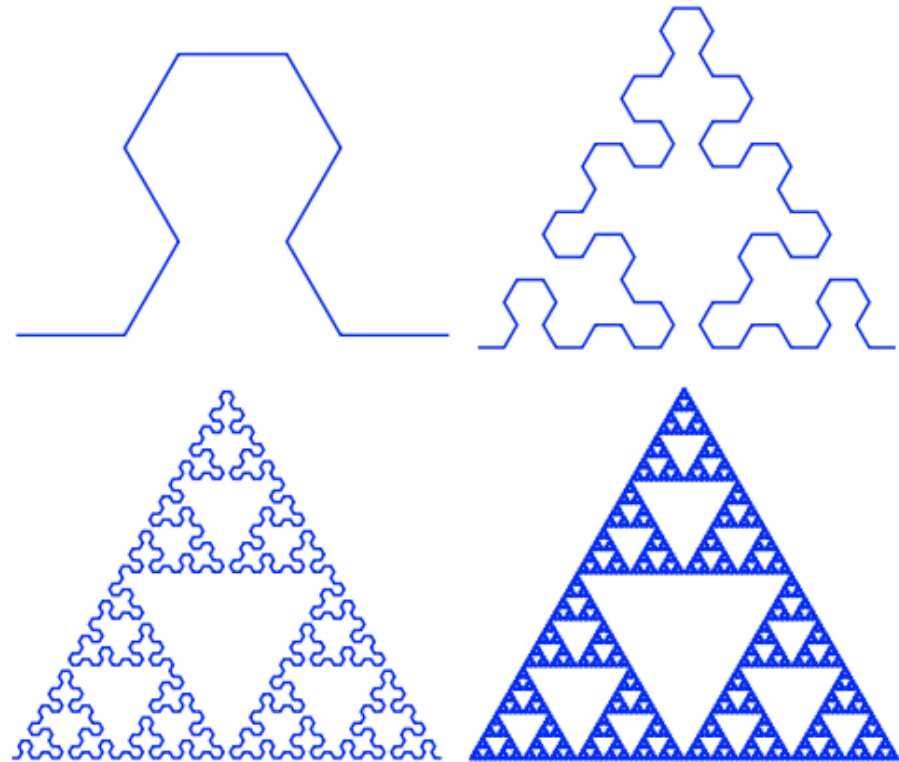
Rules:

A  $\rightarrow$  B-A-B

B  $\rightarrow$  A+B+A

Start Symbol: A

A and B both mean "draw forward",  
+ means "turn to the left  $60^\circ$ ",  
- means "turn to the right  $60^\circ$ "



# RECURSIVE GRAMMAR RULES



34

- A rule is **recursive** iff it is  $X \rightarrow w_1 Y w_2$ , where:  
 $Y \Rightarrow^* w_3 X w_4$  for some  $w_1, w_2, w_3$ , and  $w_4$  are in  $V^*$ .
- A grammar is recursive iff it contains at least one recursive rule.
- Examples:  $S \rightarrow (S)$

# RECURSIVE GRAMMAR RULES



35

- A rule is **recursive** iff it is  $X \rightarrow w_1 Y w_2$ , where:  
 $Y \Rightarrow^* w_3 X w_4$  for some  $w_1, w_2, w_3$ , and  $w_4$  are in  $V^*$ .
- A grammar is recursive iff it contains at least one recursive rule.
- Examples:  $S \rightarrow (S)$                        $S \rightarrow (T)$



# RECURSIVE GRAMMAR RULES

- A rule is **recursive** iff it is  $X \rightarrow w_1 Yw_2$ , where:  
 $Y \Rightarrow^* w_3 Xw_4$  for some  $w_1, w_2, w_3$ , and  $w_4$  are in  $V^*$ .
- A grammar is recursive iff it contains at least one recursive rule.
- Examples:



# SELF-EMBEDDING GRAMMAR RULES

- A rule in a grammar  $G$  is **self-embedding** iff it is :  
$$X \rightarrow w_1 Y w_2, \text{ where } Y \Rightarrow^* w_3 X w_4 \text{ and}$$
$$\text{both } w_1 w_3 \text{ and } w_4 w_2 \text{ are in } \Sigma^+.$$
- A grammar is self-embedding iff it contains at least one self-embedding rule.
- Example:  $S \rightarrow aSa$  is self-embedding
- Self-embedding grammars are able to define languages like  $Bal$ ,  $A^n B^n$  and of the form  $uv^i xy^i z$

# RECURSIVE AND SELF-EMBEDDING GRAMMAR RULES



38

- A rule in a grammar  $G$  is **self-embedding** iff it is :  
 $X \rightarrow w_1 Y w_2$ , where  $Y \Rightarrow^* w_3 X w_4$  and  
both  $w_1 w_3$  and  $w_4 w_2$  are in  $\Sigma^+$ .
- A grammar is self-embedding iff it contains at least one self-embedding rule.
- Example:

$S \rightarrow a S a$	is self-embedding
$S \rightarrow a S$	is recursive but not self-embedding
$S \rightarrow a T$	
$T \rightarrow S a$	is self-embedding

(we require that a nonempty string be generated on each side of the nested X)

# RECURSIVE AND SELF-EMBEDDING GRAMMAR RULES



39

- If a grammar  $G$  is **not** self-embedding then  $L(G)$  is regular.
- A grammar  $G$  is self-embedding does not guarantee that  $L(G)$  isn't regular. Another different grammar  $G'$  that also defines  $L(G)$  is not self embedding.  
e.g.:  $G_1 = (\{S, a\}, \{a\}, \{S \rightarrow \varepsilon, S \rightarrow a, S \rightarrow aSa\}, S)$
- If a language  $L$  has the property that every grammar that defines it is self-embedding, then  $L$  is **not** regular.

$$\text{PalEven} = \{ww^R : w \in \{a, b\}^*\}$$



$$\text{PalEven} = \{ww^R : w \in \{a, b\}^*\}$$

$G = \{\{S, a, b\}, \{a, b\}, R, S\}$ , where:

$$R = \{ \begin{array}{l} S \rightarrow aSa \\ S \rightarrow bSb \\ S \rightarrow \varepsilon \end{array} \}.$$

# EQUAL NUMBERS OF a's AND b's

42

Let  $L = \{w \in \{a, b\}^*: \#_a(w) = \#_b(w)\}$ .

# EQUAL NUMBERS OF a's AND b's

Let  $L = \{w \in \{a, b\}^*: \#_a(w) = \#_b(w)\}$ .

$G = \{\{S, a, b\}, \{a, b\}, R, S\}$ , where:

$$R = \{ \begin{array}{l} S \rightarrow aSb \\ S \rightarrow bSa \\ S \rightarrow SS \\ S \rightarrow \varepsilon \end{array} \}.$$

# ARITHMETIC EXPRESSIONS

$$\begin{aligned} G &= (V, \Sigma, R, E), \text{ where} \\ V &= \{+, *, (, ), \text{id}, E\}, \\ \Sigma &= \{+, *, (, ), \text{id}\}, \\ R &= \{ \\ &\quad E \rightarrow E + E \\ &\quad E \rightarrow E * E \\ &\quad E \rightarrow (E) \\ &\quad E \rightarrow \text{id} \} \end{aligned}$$

# BACKUS NAUR FORM (BNF)

A notation for writing practical context-free grammars

- The symbol  $|$  should be read as “or”.

Example:  $S \rightarrow aSb \mid bSa \mid SS \mid \varepsilon$

- Allow a nonterminal symbol to be any sequence of characters surrounded by angle brackets.

Examples of nonterminals:

$\langle \text{program} \rangle$

$\langle \text{variable} \rangle$

# BNF for a Java Fragment

46

```
<block> ::= {<stmt-list>} | {}  
<stmt-list> ::= <stmt> | <stmt-list> <stmt>  
<stmt> ::= <block> | while (<cond>) <stmt> |  
           if (<cond>) <stmt> |  
           do <stmt> while (<cond>); |  
           <assignment-stmt>; |  
           return | return <expression> |  
           <method-invocation>;
```



$S \rightarrow NP VP$

$NP \rightarrow \text{the } Nominal \mid \text{a } Nominal \mid Nominal \mid$   
 $ProperNoun \mid NP PP$

$Nominal \rightarrow N \mid Adjs N$

$N \rightarrow \text{cat} \mid \text{dogs} \mid \text{bear} \mid \text{girl} \mid \text{chocolate} \mid \text{rifle}$

$ProperNoun \rightarrow \text{Chris} \mid \text{Fluffy}$

$Adjs \rightarrow Adj Adjs \mid Adj$

$Adj \rightarrow \text{young} \mid \text{older} \mid \text{smart}$

$VP \rightarrow V \mid V NP \mid VP PP$

$V \rightarrow \text{like} \mid \text{likes} \mid \text{thinks} \mid \text{shots} \mid \text{smells}$

$PP \rightarrow Prep NP$

$Prep \rightarrow \text{with}$

# DESIGNING CONTEXT-FREE GRAMMARS

48

- Generate related regions together.

$$A^n B^n$$

- Generate concatenated regions:

$$A \rightarrow BC$$

- Generate outside in:

$$A \rightarrow aAb$$



# CONCATENATING INDEPENDENT LANGUAGES

49

Let  $L = \{a^m b^n c^m : n, m \geq 0\}$ .

The  $c^m$  portion of any string in  $L$  is completely independent of the  $a^m b^n$  portion, so we should generate the two portions separately and concatenate them together.

# CONCATENATING INDEPENDENT LANGUAGES

50

Let  $L = \{a^m b^n c^m : n, m \geq 0\}$ .

The  $c^m$  portion of any string in  $L$  is completely independent of the  $a^n b^n$  portion, so we should generate the two portions separately and concatenate them together.

$G = (\{S, N, C, a, b, c\}, \{a, b, c\}, R, S)$  where:

$$R = \left\{ \begin{array}{ll} S \rightarrow NC & /* \text{Generate two independent portions} */ \\ N \rightarrow aNb & /* \text{Generate } a^n b^n \text{ portion, from outside in} */ \\ N \rightarrow \varepsilon & \\ C \rightarrow cC & /* \text{Generate } c^m \text{ portion} */ \\ C \rightarrow \varepsilon & \end{array} \right\}.$$

$$L = \{a^{n_1}b^{n_1}a^{n_2}b^{n_2}..a^{n_k}b^{n_k} : k \geq 0 \text{ and } \forall i (n_i \geq 0)\}$$

Examples of strings in  $L$ :  $\varepsilon$ , abab, aabbbaabbbbabab

Note that  $L = \{a^n b^n : n \geq 0\}^*$ .

$$L = \{a^{n_1}b^{n_1}a^{n_2}b^{n_2}..a^{n_k}b^{n_k} : k \geq 0 \text{ and } \forall i (n_i \geq 0)\}$$

Examples of strings in  $L$ :  $\varepsilon$ , abab, aabbbaabbbbabab

Note that  $L = \{a^n b^n : n \geq 0\}^*$ .

$G = (\{S, M, a, b\}, \{a, b\}, R, S)$  where:

$$R = \{ \begin{array}{l} S \rightarrow MS \\ S \rightarrow \varepsilon \\ M \rightarrow aMb \\ M \rightarrow \varepsilon \end{array} \}.$$

# UNEQUAL a's AND b's

$$L = \{a^n b^m : n \neq m\}$$

$$G = (V, \Sigma, R, S), \text{ where}$$
$$V = \{a, b, S, \quad \},$$
$$\Sigma = \{a, b\},$$
$$R =$$

# UNEQUAL a's AND b's

$$L = \{a^n b^m : n \neq m\}$$

$$G = (V, \Sigma, R, S), \text{ where}$$
$$V = \{a, b, S, A, B\},$$
$$\Sigma = \{a, b\},$$
$$R =$$

$S \rightarrow A$	/* more a's than b's
$S \rightarrow B$	/* more b's than a's
$A \rightarrow a$	/* at least one extra a generated
$A \rightarrow aA$	
$A \rightarrow aAb$	
$B \rightarrow b$	/* at least one extra b generated
$B \rightarrow Bb$	
$B \rightarrow aBb$	

# SIMPLIFYING CONTEXT-FREE GRAMMARS



55

$G = (\{S, A, B, C, D, a, b\}, \{a, b\}, R, S)$ , where

$R =$

$$\begin{aligned} \{ & S \rightarrow AB \mid AC \\ & A \rightarrow aAb \mid \varepsilon \\ & B \rightarrow aA \\ & C \rightarrow bCa \\ & D \rightarrow AB \} \end{aligned}$$



*removeunproductive*( $G$ : CFG) =

1.  $G' = G$ .
2. Mark every nonterminal symbol in  $G'$  as unproductive.
3. Mark every terminal symbol in  $G'$  as productive.
4. Until one entire pass has been made without any new symbol being marked do:
  - For each rule  $X \rightarrow \alpha$  in  $R$  do:
    - If every symbol in  $\alpha$  has been marked as productive and  $X$  has not yet been marked as productive then:
      - Mark  $X$  as productive.
5. Remove from  $G'$  every unproductive symbol.
6. Remove from  $G'$  every rule that contains an unproductive symbol.
7. Return  $G'$ .





*removeunreachable*( $G$ : CFG) =

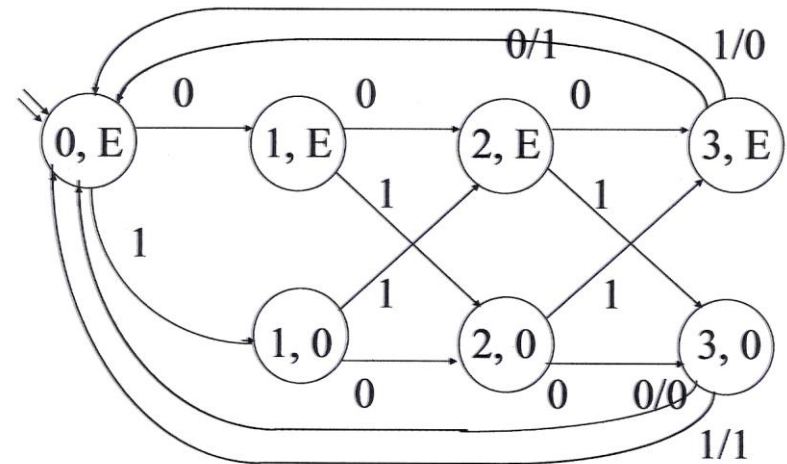
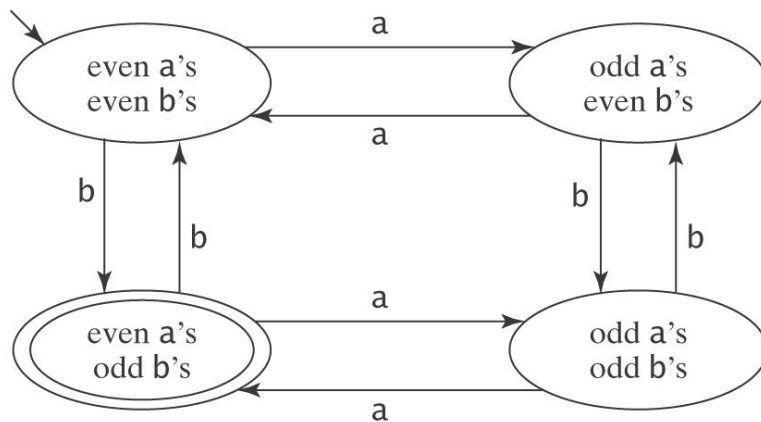
1.  $G' = G$ .
2. Mark  $S$  as reachable.
3. Mark every other nonterminal symbol as unreachable.
4. Until one entire pass has been made without any new symbol being marked do:
  - For each rule  $X \rightarrow \alpha A \beta$  (where  $A \in V - \Sigma$ ) in  $R$  do:
    - If  $X$  has been marked as reachable and  $A$  has not then:
      - Mark  $A$  as reachable.
5. Remove from  $G'$  every unreachable symbol.
6. Remove from  $G'$  every rule with an unreachable symbol on the left-hand side.
7. Return  $G'$ .

# REGULAR VS. CONTEXT-FREE LANGUAGES

58

Regular languages:

We care about recognizing patterns and taking appropriate actions.

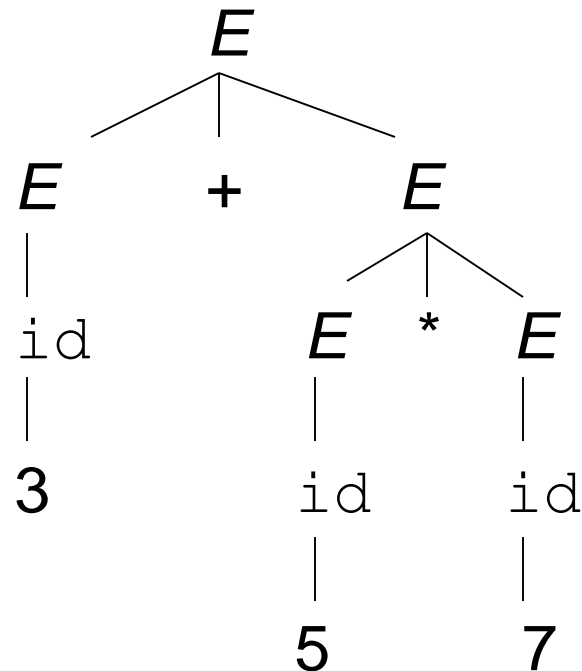


# REGULAR VS. CONTEXT-FREE LANGUAGES

59

Context free languages:

We care about structure.



# DERIVATIONS



60

To capture structure, we must capture the path we took through the grammar. **Derivations** do that.

Example:

$$\begin{aligned} S &\rightarrow \varepsilon \\ S &\rightarrow SS \\ S &\rightarrow (S) \end{aligned}$$

$$\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & & & \\ S \Rightarrow & SS \Rightarrow & (S)S \Rightarrow & ((S))S \Rightarrow & (()S \Rightarrow & (())(S) \Rightarrow & (())( & & \\ S \Rightarrow & SS \Rightarrow & (S)S \Rightarrow & ((S))S \Rightarrow & ((S))(S) \Rightarrow & (())(S) \Rightarrow & (())( & & \\ & 1 & 2 & 3 & 5 & 4 & 6 & & \end{array}$$

But the order of rule application doesn't matter.

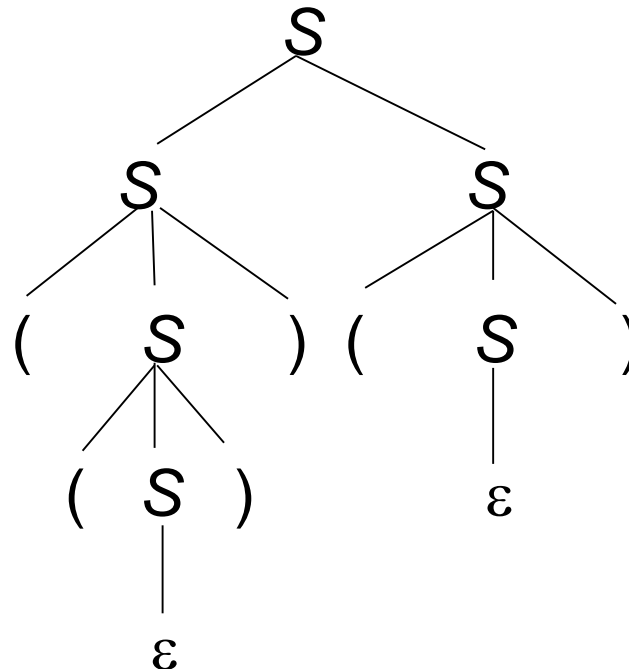
# DERIVATIONS



61

Parse trees capture essential structure:

$$\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())() \\ S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow ((S))(S) \Rightarrow (())(S) \Rightarrow (())() \\ 1 & 2 & 3 & 5 & 4 & 6 \end{array}$$



# PARSE TREES

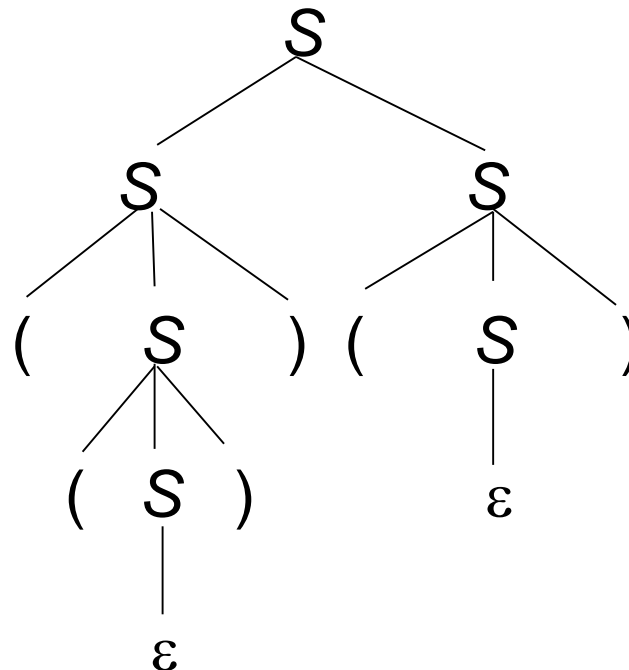


62

A parse tree, derived by a grammar  $G = (V, \Sigma, R, S)$ , is a rooted, ordered tree in which:

- Every leaf node is labeled with an element of  $\Sigma \cup \{\varepsilon\}$ ,
- The root node is labeled  $S$ ,
- Every other node is labeled with some element of:  
 $V - \Sigma$ , and
- If  $m$  is a nonleaf node labeled  $X$  and the children of  $m$  are labeled  $x_1, x_2, \dots, x_n$ , then  $R$  contains the rule  
 $X \rightarrow x_1, x_2, \dots, x_n$ .

## 63

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$$


**Left-most derivation:** At each step of derivation, the leftmost nonterminal in the working string is chose for expansion.

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$$

**Right-most derivation:** At each step of derivation, the rightmost nonterminal in the working string is chose for expansion.

$$S \Rightarrow SS \Rightarrow S(S) \Rightarrow S() \Rightarrow (S)() \Rightarrow ((S))() \Rightarrow (())()$$



# Branching Factor

65

**Branching factor** of a grammar  $G$  to be length (the number of symbols) of the longest right-hand side of any rule in  $G$

**Branching factor** of any parse tree generated by  $G$  is less than or equal to the branching factor of  $G$ .



Because parse trees matter, it makes sense, given a grammar  $G$ , to distinguish between:

- $G$ 's ***weak generative capacity***, defined to be the set of strings,  $L(G)$ , that  $G$  generates, and
- $G$ 's ***strong generative capacity***, defined to be the set of parse trees that  $G$  generates.

# AMBIGUITY

67

A grammar is ***ambiguous*** iff there is at least one string in  $L(G)$  for which  $G$  produces more than one parse tree.

For most applications of context-free grammars, this is a problem.

# ARITHMETIC EXPRESSIONS

68

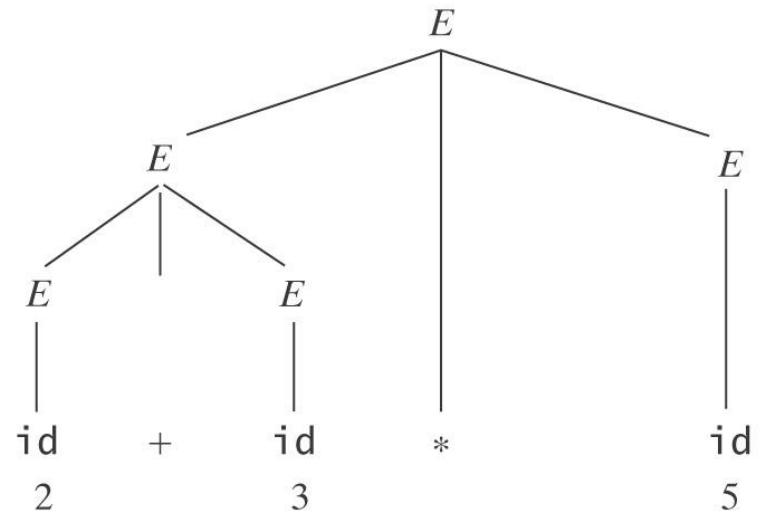
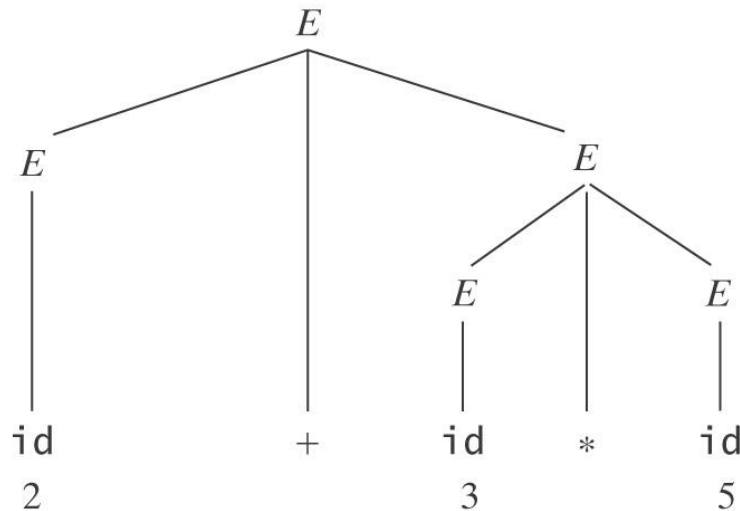
$$\begin{aligned} G &= (V, \Sigma, R, E), \text{ where} \\ V &= \{+, *, (, ), \text{id}, E\}, \\ \Sigma &= \{+, *, (, ), \text{id}\}, \\ R &= \{ E \rightarrow E + E \\ &\quad E \rightarrow E * E \\ &\quad E \rightarrow (E) \\ &\quad E \rightarrow \text{id} \} \end{aligned}$$

# ARITHMETIC EXPRESSIONS

69

$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow (E)$$
$$E \rightarrow \text{id}$$

2 + 3 \* 5



# AMBIGUITY

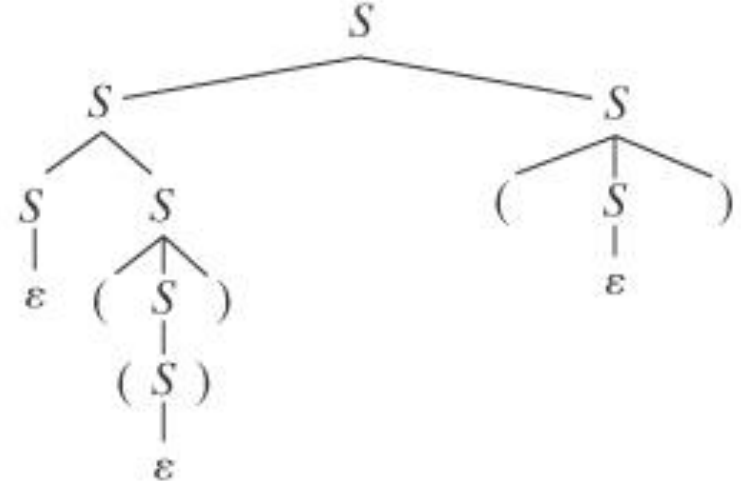
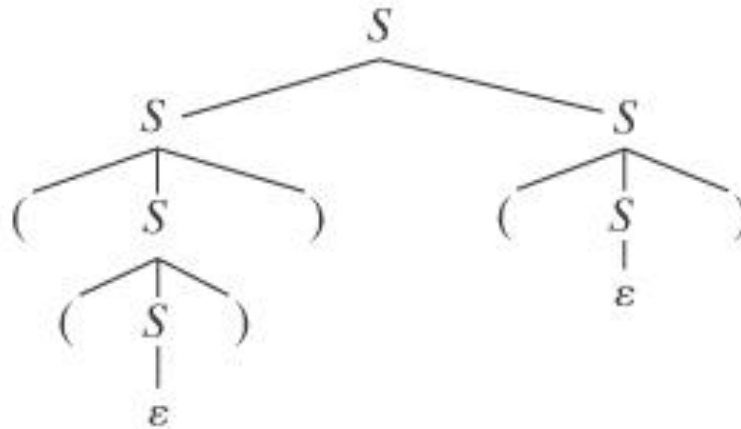
70

$$S \rightarrow \varepsilon$$

$$S \rightarrow SS$$

$$S \rightarrow (S)$$

( ( ) ) ( )



Even a Very Simple Grammar Can be Highly Ambiguous

# INHERENT AMBIGUITY

Some languages have the property that every grammar for them is ambiguous. We call such languages *inherently ambiguous*.

Example:

$$\begin{aligned} L &= \{a^i b^j c^k : i, j, k \geq 0, i=j \text{ or } j=k\} \\ &= \{a^n b^n c^m : n, m \geq 0\} \cup \{a^n b^m c^m : n, m \geq 0\}. \end{aligned}$$

# INHERENT AMBIGUITY

72

$$L = \{a^n b^n c^m : n, m \geq 0\} \cup \{a^n b^m c^m : n, m \geq 0\}.$$

One grammar for  $L$  has the rules:

$$S \rightarrow S_1 \mid S_2$$

$$\begin{array}{ll} S_1 \rightarrow S_1 c \mid A & /* \text{Generate all strings in } \{a^n b^n c^m\}. \\ A \rightarrow aAb \mid \varepsilon \end{array}$$

$$\begin{array}{ll} S_2 \rightarrow aS_2 \mid B & /* \text{Generate all strings in } \{a^n b^m c^m\}. \\ B \rightarrow bBc \mid \varepsilon \end{array}$$

Consider any string of the form  $a^n b^n c^n$ .

$L$  is inherently ambiguous.



Both of the following problems are undecidable:

- Given a context-free grammar  $G$ , is  $G$  ambiguous?
- Given a context-free language  $L$ , is  $L$  inherently ambiguous?

# REDUCING AMBIGUITY

We can get rid of:

- $\varepsilon$  rules like  $S \rightarrow \varepsilon$ ,
- recursive rules with symmetric right-hand sides, e.g.,

$$S \rightarrow SS$$

$$E \rightarrow E + E$$

- rule sets that lead to ambiguous attachment of optional postfixes.

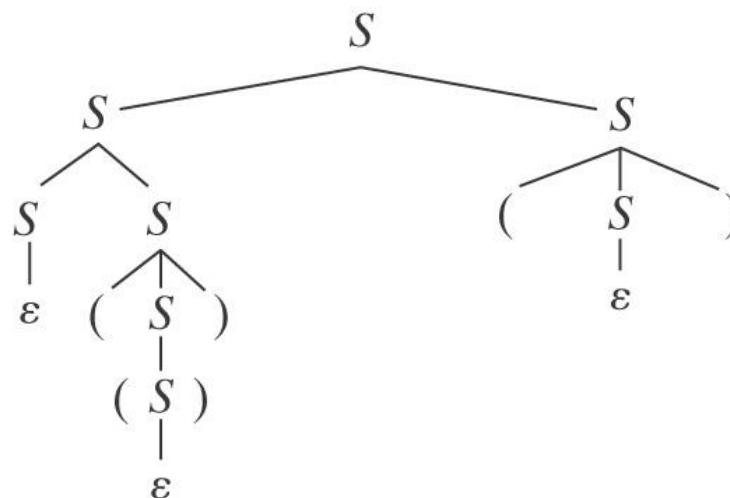
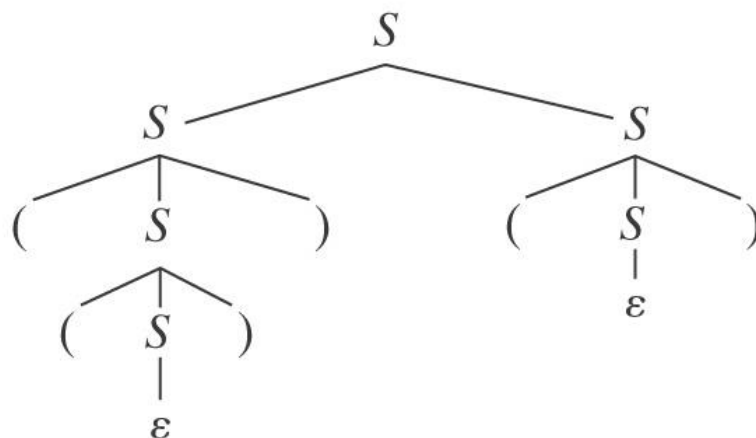
# REDUCING AMBIGUITY

75

$$S \rightarrow \varepsilon$$

$$S \rightarrow SS$$

$$S \rightarrow (S)$$



# REDUCING AMBIGUITY

76

A different grammar for the language of balanced parentheses:

$$S^* \rightarrow \varepsilon$$

$$S^* \rightarrow S$$

$$S \rightarrow SS$$

$$S \rightarrow (S)$$

$$S \rightarrow ()$$

The grammar is still ambiguous. Need to eliminate symmetric recursive rule  $S \rightarrow SS$ .

[See: Page 227-230]

# References

- ❑ **Automata, Computability and Complexity. Theory and Applications**
  - By Elaine Rich
- ❑ Chapter 11:
  - Page : 203-224.