# Automata Theory

## *CS411-2015F-08*

## *Context-Free Grammars*

David Galles

Department of Computer Science
University of San Francisco

# 08-0: Context-Free Grammars

- Set of Terminals ($\Sigma$)
- Set of Non-Terminals
- Set of Rules, each of the form: $<$Non-Terminal$>$ $\rightarrow$ $<$Terminals & Non-Terminals$>$
- Special Non-Terminal – Initial Symbol

# 08-1: Generating Strings with CFGs

- Start with the initial symbol
- Repeat:
  - Pick any non-terminal in the string
  - Replace that non-terminal with the right-hand side of some rule that has that non-terminal as a left-hand side

  Until all elements in the string are terminals

# 08-2: CFG Example

$S \rightarrow aS$

$S \rightarrow Bb$

$B \rightarrow cB$

$B \rightarrow \epsilon$

Generating a string:

| | |
|---|---|
| $S$ | replace $S$ with $aS$ |
| $aS$ | replace $S$ wtih $Bb$ |
| $aBb$ | replace $B$ wtih $cB$ |
| $acBb$ | replace $B$ wtih $\epsilon$ |
| $acb$ | Final String |

# 08-3: CFG Example

$S \rightarrow aS$

$S \rightarrow Bb$

$B \rightarrow cB$

$B \rightarrow \epsilon$

Generating a string:

| | |
|---|---|
| $S$ | replace $S$ with $aS$ |
| $aS$ | replace $S$ wtih $aS$ |
| $aaS$ | replace $S$ wtih $Bb$ |
| $aaBb$ | replace $B$ wtih $cB$ |
| $aacBb$ | replace $B$ wtih $cB$ |
| $aaccBb$ | replace $B$ wtih $\epsilon$ |
| $aaccb$ | Final String |

# 08-4: CFG Example

$S \rightarrow aS$

$S \rightarrow Bb$

$B \rightarrow cB$

$B \rightarrow \epsilon$

Regular Expression equivalent to this CFG:

# 08-5: CFG Example

$S \rightarrow aS$

$S \rightarrow Bb$

$B \rightarrow cB$

$B \rightarrow \epsilon$

Regular Expression equivalent to this CFG:

$a^*c^*b$

# 08-6: CFG Example

CFG for $L = \{0^n 1^n : n > 0\}$

# 08-7: CFG Example

CFG for $L = \{0^n 1^n : n > 0\}$

$S \rightarrow 0S1$ or $S \rightarrow 0S1|01$

$S \rightarrow 01$

(note – can write:

$A \rightarrow \alpha$

$A \rightarrow \beta$

as

$A \rightarrow \alpha|\beta$ )

(examples: 01, 0011, 000111)

# 08-8: CFG Formal Definition

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

- $V$ = Set of symbols, both terminals & non-terminals
- $\Sigma \subset V$ set of terminals (alphabet for the language being described)
- $R \subset ((V - \Sigma) \times V^*)$ Finite set of rules
- $S \in (V - \Sigma)$ Start symbol

# 08-9: CFG Formal Definition

Example:

$S \rightarrow 0S1$

$S \rightarrow 01$

Set theory Definition:

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

- $V = \{S, 0, 1\}$
- $\Sigma \subset V = \{0, 1\}$
- $R \subset ((V - \Sigma) \times V^*) = \{(S, 0S0), (S, 01)\}$
- $S \in (V - \Sigma) = S$

# 08-10: Derivation

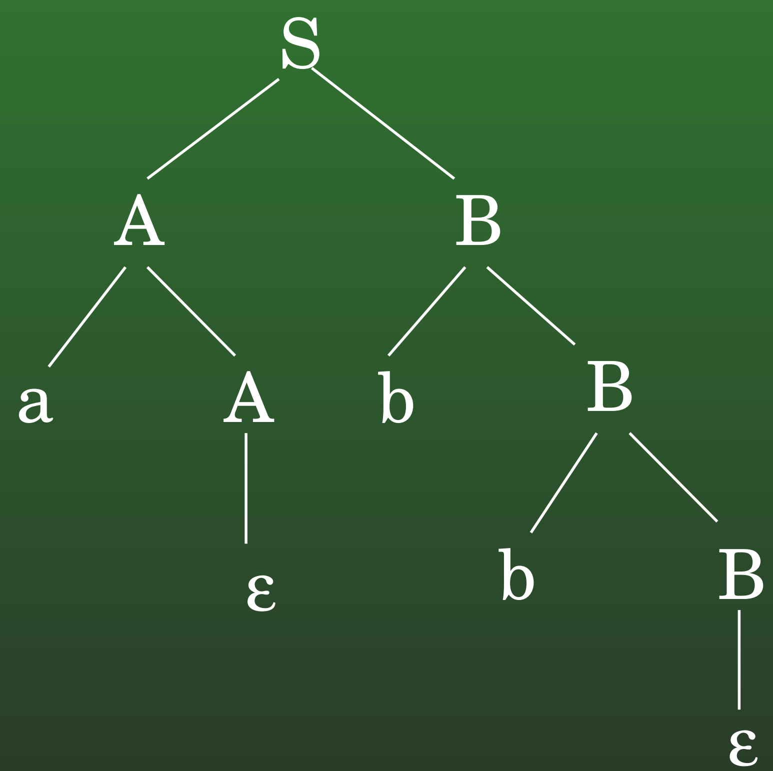
A *Derivation* is a listing of how a string is generated – showing what the string looks like after every replacement.

$$S \rightarrow AB$$
$$A \rightarrow aA|\epsilon$$
$$B \rightarrow bB|\epsilon$$

$$\begin{aligned} S &\Rightarrow AB \\ &\Rightarrow aAB \\ &\Rightarrow aAbB \\ &\Rightarrow abB \\ &\Rightarrow abbB \\ &\Rightarrow abb \end{aligned}$$

# 08-11: Parse Tree

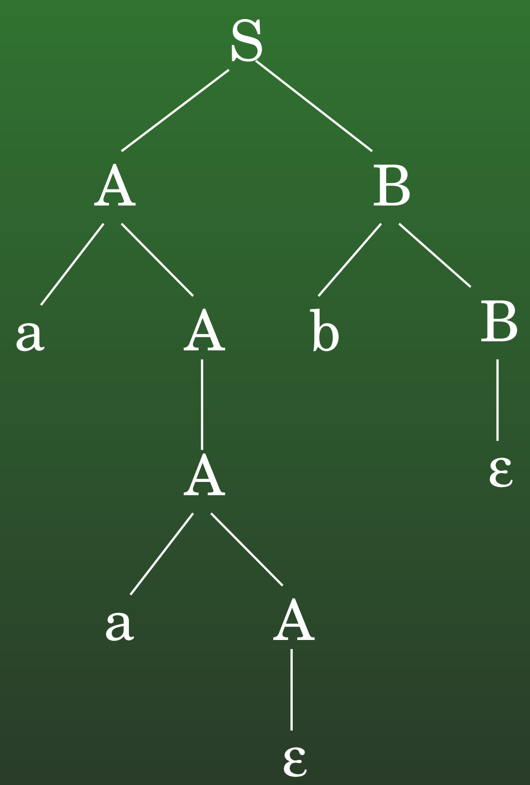
A *Parse Tree* is a graphical representation of a derivation.




$$
\begin{aligned}
S &\Rightarrow AB \\
&\Rightarrow aAB \\
&\Rightarrow aAbB \\
&\Rightarrow abB \\
&\Rightarrow abbB \\
&\Rightarrow abb
\end{aligned}
$$

# 08-12: Parse Tree

A *Parse Tree* is a graphical representation of a derivation.




$$\begin{aligned} S &\Rightarrow AB \\ &\Rightarrow AbB \\ &\Rightarrow aAbB \\ &\Rightarrow aaAbB \\ &\Rightarrow aaAb \\ &\Rightarrow aab \end{aligned}$$

# 08-13: Fun with CFGs

- Create a Context-Free Grammar for all strings over {a,b} which contain the substring "aba"

# 08-14: Fun with CFGs

- Create a Context-Free Grammar for all strings over {a,b} which contain the substring “aba”

$S \rightarrow A\text{aba}A$

$A \rightarrow \text{a}A$

$A \rightarrow \text{b}A$

$A \rightarrow \epsilon$

- Give a parse tree for the string: bbabaa

# 08-15: Fun with CFGs

- Create a Context-Free Grammar for all strings over {a,b} that begin or end with the substring bba (inclusive or)

# 08-16: Fun with CFGs

- Create a Context-Free Grammar for all strings over {a,b} that begin or end with the substring bba (inclusive or)

$S \rightarrow \text{bba}A$

$S \rightarrow A\text{bba}$

$A \rightarrow \text{b}A$

$A \rightarrow \text{a}A$

$A \rightarrow \epsilon$

# 08-17: $L_{CFG}$

The Context-Free Languages, $L_{CFG}$, is the set of all languages that can be described by some CFG:

- $L_{CFG} = \{L : \exists$ CFG $G \wedge L[G] = L\}$

We already know $L_{CFG} \not\subseteq L_{REG}$ (why)?

- $L_{REG} \subset L_{CFG}$ ?

# 08-18: $L_{REG} \subseteq L_{CFG}$

We will prove $L_{REG} \subseteq L_{CFG}$ in two different ways:

- Prove by induction that, given any regular expression $r$, we create a CFG $G$ such that $L[G] = L[r]$
- Given any NFA $M$, we create a CFG $G$ such that $L[G] = L[M]$

# 08-19: $L_{REG} \subseteq L_{CFG}$

- To Prove: Given any regular expression $r$, we can create a CFG $G$ such that $L[G] = L[r]$
- By induction on the structure of $r$

# 08-20: $L_{REG} \subseteq L_{CFG}$

Base Cases:

- $r$ = a, a $\in \Sigma$

# 08-21: $L_{REG} \subseteq L_{CFG}$

Base Cases:

- $r$ = a, a $\in \Sigma$

$S \rightarrow$ a

## 08-22: $L_{REG} \subseteq L_{CFG}$

Base Cases:

- $r = \epsilon$

# 08-23: $L_{REG} \subseteq L_{CFG}$

Base Cases:

- $r = \epsilon$

$S \rightarrow \epsilon$

# 08-24: $L_{REG} \subseteq L_{CFG}$

Base Cases:

- $r = \emptyset$

# 08-25: $L_{REG} \subseteq L_{CFG}$

Base Cases:

- $r = \emptyset$

$S \rightarrow SS$

# 08-26: $L_{REG} \subseteq L_{CFG}$

Recursive Cases:

- $r = (r_1 r_2)$

$L[G_1] = L[r_1]$, Start symbol of $G_1 = S_1$
$L[G_2] = L[r_2]$, Start symbol of $G_2 = S_2$

# 08-27: $L_{REG} \subseteq L_{CFG}$

Recursive Cases:

- $r = (r_1 r_2)$

$L[G_1] = L[r_1]$, Start symbol of $G_1 = S_1$
$L[G_2] = L[r_2]$, Start symbol of $G_2 = S_2$

$G$ = all rules from $G_1$ and $G_2$, plus plus new non-terminal $S$, and new rule:

$S \rightarrow S_1 S_2$

New start symbol $S$

# 08-28: $L_{REG} \subseteq L_{CFG}$

Recursive Cases:

- $r = (r_1 + r_2)$

$L[G_1] = L[r_1]$, Start symbol of $G_1 = S_1$
$L[G_2] = L[r_2]$, Start symbol of $G_2 = S_2$

# 08-29: $L_{REG} \subseteq L_{CFG}$

Recursive Cases:

- $r = (r_1 + r_2)$

$L[G_1] = L[r_1]$, Start symbol of $G_1 = S_1$
$L[G_2] = L[r_2]$, Start symbol of $G_2 = S_2$

$G$ = all rules from $G_1$ and $G_2$, plus new non-terminal $S$, and new rules:

$S \rightarrow S_1$
$S \rightarrow S_2$

Start symbol = $S$

# 08-30: $L_{REG} \subseteq L_{CFG}$

Recursive Cases:

- $r = (r_1^*)$

$L[G_1] = L[r_1]$, Start symbol of $G_1 = S_1$

# 08-31: $L_{REG} \subseteq L_{CFG}$

Recursive Cases:

- $r = (r_1^*)$

$L[G_1] = L[r_1]$, Start symbol of $G_1 = S_1$

$G$ = all rules from $G_1$, plus new non-terminal $S$, and new rules:

$S \rightarrow S_1 S$
$S \rightarrow \epsilon$

Start symbol = $S$

(Example)

# 08-32: $L_{REG} \subseteq L_{CFG}$ II

- Given any NFA
  - $M = (K, \Sigma, \Delta, s, F)$
- Create a grammar
  - $G = (V, \Sigma, R, S)$ such that $L[G] = L[M]$
- Idea: Derivations like “backward NFA configurations”, showing past instead of future
  - Example for all strings over {a, b} that contain aa, not bb

# 08-33: $L_{REG} \subseteq L_{CFG}$ II

- $M = (K, \Sigma, \Delta, s, F)$
- $G = (V, \Sigma', R, S)$
    - $V$
    - $\Sigma'$
    - $R$
    - $S$

## 08-34: $L_{REG} \subseteq L_{CFG}$ II

- $M = (K, \Sigma, \Delta, s, F)$
- $G = (V, \Sigma', R, S)$
  - $V = K \bigcup \Sigma$
  - $\Sigma' = \Sigma$
  - $R = \{(q_1 \rightarrow aq_2) : q_1, q_2 \in K$ (and $V$),
    $a \in \Sigma, ((q_1, a), q_2) \in \Delta\} \cup$
    $\{(q \rightarrow \epsilon) : q \in F\}$
  - $S = s$

(Example)

# CFG – Ambiguity

- A CFG is *ambiguous* if there exists at least one string generated by the grammar that has > 1 different parse tree
- Previous CFG is ambiguous (examples)

  $S \rightarrow A\text{aba}A$

  $A \rightarrow \text{a}A$

  $A \rightarrow \text{b}A$

  $A \rightarrow \epsilon$

# 08-36: CFG – Ambiguity

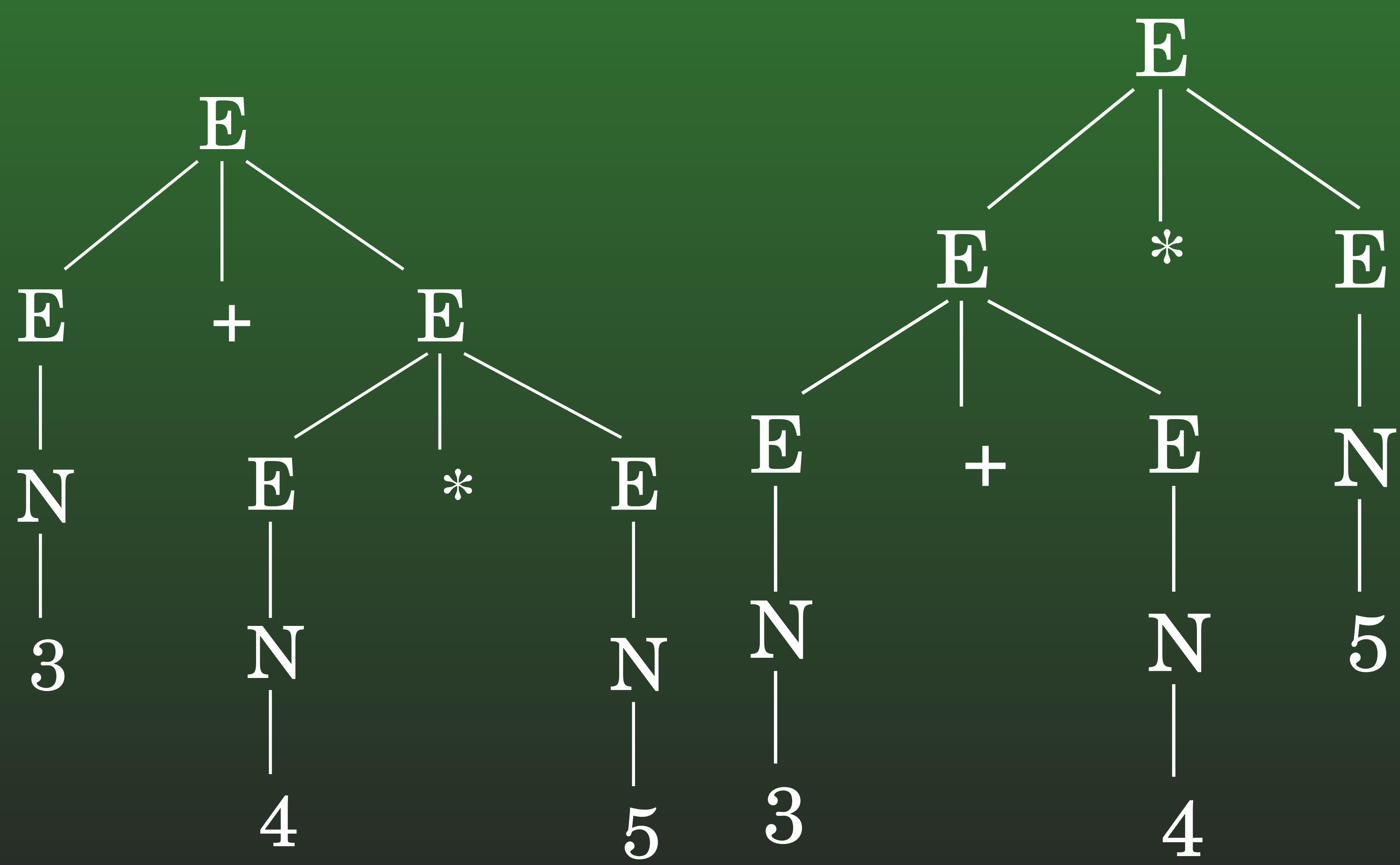
- Consider the following CFG:

$$E \rightarrow E + E | E - E | E * E | N$$
$$N \rightarrow 0|1|2|3|4|5|6|7|8|9$$

- Is this CFG ambiguous?
- Why is this a problem?

# CFG – Ambiguity

$$E \rightarrow E + E | E - E | E * E | N$$
$$N \rightarrow 0|1|2|3|4|5|6|7|8|9$$



# 08-38: CFG – Ambiguity

$$E \rightarrow E + E | E - E | E * E | N$$
$$N \rightarrow 0|1|2|3|4|5|6|7|8|9$$

- If all we care about is removing ambiguity, there is a (relatively) easy way to make this unambiguous (make all operators right-associative)

# 08-39: CFG – Ambiguity

$$E \rightarrow E + E | E - E | E * E | N$$
$$N \rightarrow 0|1|2|3|4|5|6|7|8|9$$

Non-ambiguous:

$$E \rightarrow N | N + E | N - E | N * E$$
$$N \rightarrow 0|1|2|3|4|5|6|7|8|9$$

- If we were writing a compiler, would this be a good CFG?
- How can we get correct associativity

# 08-40: CFG – Ambiguity

- Ambiguous:

  $E \rightarrow E + E | E - E | E * E | N$

  $N \rightarrow 0|1|2|3|4|5|6|7|8|9$

- Unambiguous:

  $E \rightarrow E + T | E - T | T$

  $T \rightarrow T * N | N$

  $N \rightarrow 0|1|2|3|4|5|6|7|8|9$

Can add parentheses, other operators, etc. (More in Compilers)

# Fun with CFGs

- Create a CFG for all strings over {(,)} that form balanced parenthesis
  - ()
  - ()()
  - (()())((()()()))
  - ((((()))))

# 08-42: Fun with CFGs

- Create a CFG for all strings over {(,)} that form balanced parenthesis

$$S \rightarrow (S)$$
$$S \rightarrow SS$$
$$S \rightarrow \epsilon$$

- Is this grammar ambiguous?

# 08-43: Fun with CFGs

- Create a CFG for all strings over {(,)} that form balanced parenthesis

  $S \rightarrow (S)$

  $S \rightarrow SS$

  $S \rightarrow \epsilon$

- Is this grammar ambiguous?
  - YES! (examples)

# 08-44: Fun with CFGs

- Create an *unambiguous* CFG for all strings over {(,)} that form balanced parenthesis

# 08-45: Fun with CFGs

- Create an *unambiguous* CFG for all strings over {(,)} that form balanced parenthesis

$S \rightarrow AS$

$S \rightarrow \epsilon$

$A \rightarrow (S)$

# 08-46: Ambiguous Languages

- A language $L$ is ambiguous if all CFGs $G$ that generate it are ambiguous
- Example:
  - $L_1 = \{a^i b^i c^j d^j | i, j > 0\}$
  - $L_2 = \{a^i b^j c^j d^i | i, j > 0\}$
  - $L_3 = L_1 \cup L_2$
- $L_3$ is inherently ambiguous

(Create a CFG for $L_3$)

# 08-47: Ambiguous Languages

- $L_1 = \{a^i b^i c^j d^j | i, j > 0\}$
- $L_2 = \{a^i b^j c^j d^i | i, j > 0\}$
- $L_3 = L_1 \cup L_2$

$$
\begin{aligned}
S &\rightarrow S_1 | S_2 \\
S_1 &\rightarrow AB \\
A &\rightarrow aAb | ab \\
B &\rightarrow cBd | cd \\
S_2 &\rightarrow aS_2d | aCd \\
C &\rightarrow bCc | bc
\end{aligned}
$$

What happens when $i = j$?

# 08-48: (More) Fun with CFGs

- Create an CFG for all strings over {a, b} that have the same number of a's as b's (can be ambiguous)

# (More) Fun with CFGs

- Create an CFG for all strings over {a, b} that have the same number of a's as b's (can be ambiguous)

$$S \rightarrow aSb$$
$$S \rightarrow bSa$$
$$S \rightarrow SS$$
$$S \rightarrow \epsilon$$

# 08-50: (More) Fun with CFGs

- Create an CFG for $L = \{ww^R : w \in (a+b)^*\}$

# 08-51: (More) Fun with CFGs

- Create an CFG for $L = \{ww^R : w \in (a+b)^*\}$

$S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow \epsilon$

# 08-52: (More) Fun with CFGs

- Create an CFG for all palindromes over $\{a, b\}$. That is, create a CFG for:
  - $L = \{w : w \in (a + b)^*, w = w^R\}$

# (More) Fun with CFGs

- Create an CFG for all palindromes over $\{a, b\}$. That is, create a CFG for:
  - $L = \{w : w \in (a+b)^*, w = w^R\}$

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow \epsilon$

$S \rightarrow a$

$S \rightarrow b$

# 08-54: (More) Fun with CFGs

- Create an CFG for $L = \{a^i b^j c^k : j > i + k\}$

# 08-55: (More) Fun with CFGs

- Create an CFG for $L = \{a^i b^j c^k : j > i + k\}$

*HINT:* We may wish to break this down into 3 different langauges ...

# 08-56: (More) Fun with CFGs

- Create an CFG for $L = \{a^i b^j c^k : j > i + k\}$

$S \rightarrow ABC$

$A \rightarrow aAb$

$A \rightarrow \epsilon$

$B \rightarrow bB$

$B \rightarrow b$

$C \rightarrow bCc|\epsilon$

# 08-57: (More) Fun with CFGs

- Create an CFG for all strings over {0, 1} that have the an even number of 0's and an odd number of 1's.
  - $HINT$: It may be easier to come up with 4 CFGs – even 0's, even 1's, odd 0's odd 1's, even 0's odd 1's, odd 1's, even 0's – and combine them ...

# 08-58: (More) Fun with CFGs

- Create an CFG for all strings over {0, 1} that have the an even number of 0's and an odd number of 1's.

$S_1$ = Even 0's Even 1's

$S_2$ = Even 0's Odd 1's

$S_3$ = Odd 0's Even 1's

$S_4$ = Odd 0's Odd 1's

$$S_1 \rightarrow 0S_3|1S_2$$
$$S_2 \rightarrow 0S_4|1S_1$$
$$S_3 \rightarrow 0S_1|1S_4$$
$$S_4 \rightarrow 0S_2|1S_3$$