Context-Free Grammars

CS 271, Spring 2014

Dr. Sara Miner More

Introduction

- Before today: **regular languages**
  - DFAs, NFAs
  - Regular expression
  - But not all languages are regular…
- Today: a new class of languages
  - We’ll discuss a new way to describe a language by expressing how to generate all of its strings
  - Reference: Sipser textbook Section 2.1

Context-Free Grammars (CFGs): a formal definition

- A CFG G is a 4-tuple (V, Σ, R, S), where
  - V is a finite set called the **variables**
  - Σ is a finite set, disjoint from V, called the **terminals**
  - R is a finite set of **rules**
  - S ∈ V is the **start symbol**

  - Each rule consists of a rightward arrow, with a variable on the LHS and a sequence of variables and/or terminals on the RHS
    - Convention: Variables are usually *CAPITAL letters*, terminals are usually *lower-case letters*

Example: A CFG called G_1

- **Variables:** $V = \{ E, O \}$
- **Terminals:** $\Sigma = \{ +, *, a, b \}$
- **Rules (a.k.a. Productions):**

  $R = \{ E \to a, E \to b, E \to EOE, O \to +, O \to * \}$
- **Start Symbol:** $S = E$

To derive a string in $L(G_1)$, begin with start symbol and repeatedly apply rules until no variables remain.

Notation

- If u, v, and w are strings of variables and terminals, and $\boxed{A \rightarrow w}$ is a rule of a grammar, we say

  "uAv yields uwv", denoted by

$$uAv \boxed{\Rightarrow} uwv$$

Example: A CFG called G_1

- **Rules of G_1**

  $E \rightarrow a$

  $E \rightarrow b$

  $E \rightarrow EOE$

  $O \rightarrow +$

  $O \rightarrow *$

$E \Rightarrow b$

$E \Rightarrow EOE \Rightarrow aOE$

$\Rightarrow a+E \Rightarrow a+b$

Example: A CFG called G_1

- **Variables:** $V = \{ E, O \}$
- **Terminals:** $\Sigma = \{ +, *, a, b \}$
- **Rules (a.k.a. Productions):**

  $R = \{ E \rightarrow a, E \rightarrow b, E \rightarrow EOE, O \rightarrow +, O \rightarrow * \}$
- **Start Symbol:** $S = E$

To derive a string in $L(G_1)$, begin with start symbol and repeatedly apply rules until no variables remain.

The "language of grammar G_1", denoted $L(G_1)$, is the set of all strings over Σ that can be generated from these rules, starting from E.

Example: A CFG called G_1

- **Rules of G_1**

  E → a

  E → b

  E → EOE

  O → +

  O → *

These three rules may be rewritten on one line:

E → a | b | EOE

Equivalent way to write rules of G_1:

E → a|b|EOE

O → +|*

We often specify a CFG by writing only its rules.

Convention says that variable on LHS of first rule is start symbol; the rest of formal description can be deduced!

Construct CFG G_2 where $L(G_2) = \{ 0^n1^n \mid n \geq 1 \}$

$E \rightarrow 0B1$

$B \rightarrow 0B1 \mid \varepsilon$

$E \rightarrow 0E1 \mid 01$

Construct CFG G_2 where $L(G_2) = \{ 0^n1^n \mid n \geq 1 \}$

- Are you convinced that our construction works?
  - Check for completeness:
    Does G_2 generate ***every*** string in $\{ 0^n1^n \mid n \geq 1 \}$?
  - Check for consistency:
    Does G_2 generate ***only*** the strings in $\{ 0^n1^n \mid n \geq 1 \}$?

Construct a CFG G_3 where $L(G_3) = \{ w \mid w \text{ is a palindrome over } \{a,b\} \}$

$$S \to \varepsilon \mid aSa \mid bSb$$

$$\mid a \mid b$$

Construct a CFG G_3 where $L(G_3)$ = { w | w is a palindrome over {a,b} }

- Must have same symbol at beginning and end, so insert both within application of one rule
  - Repeat this type of rule as necessary, building up the string from the ends towards the middle

- But, what about last rule used?
  - Is length of string even or odd?
    - If even, last step replaces variable with ε
    - If odd, last step replaces variable with either a or b

Construct a CFG G_3 where $L(G_3)$ = { w | w is a palindrome over {a,b} }

- $G_3 = (V, \Sigma, R, S)$, where:
  - V = {S}
  - Σ = {a,b}
  - R = { $S \rightarrow$ aSa | bSb | ε | a | b }
  - S is the start symbol

Construct a CFG G_4 where $L(G_4) = \{\boxed{0^a1^b0^c} \mid a+c = b\}$

0110

01111000

0011

$\overset{a}{0}\overset{a}{1}\overset{c}{1}\overset{c}{0}$

$S \rightarrow FB$

$F \rightarrow 0F1 \mid \varepsilon$

$B \rightarrow 1B0 \mid \varepsilon$

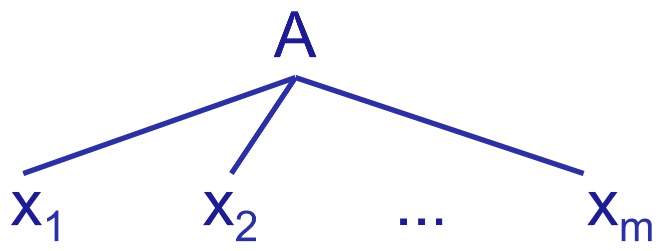
Context-Free Languages

- Definition: A language is called ***context-free*** if it is generated by a context-free grammar

- How does the class of context-free languages compare to the class of regular languages?
  - How do you know?

Parse trees

- A parse tree from a grammar G = (V, Σ, R, S) is labeled tree rooted at S where:
  - each leaf of tree is labeled with some a ∈ Σ,
  - each non-leaf of tree is labeled with some a ∈ V, and
  - if tree contains subtree:



then $A \rightarrow x_1 x_2 \ldots x_m \in R$

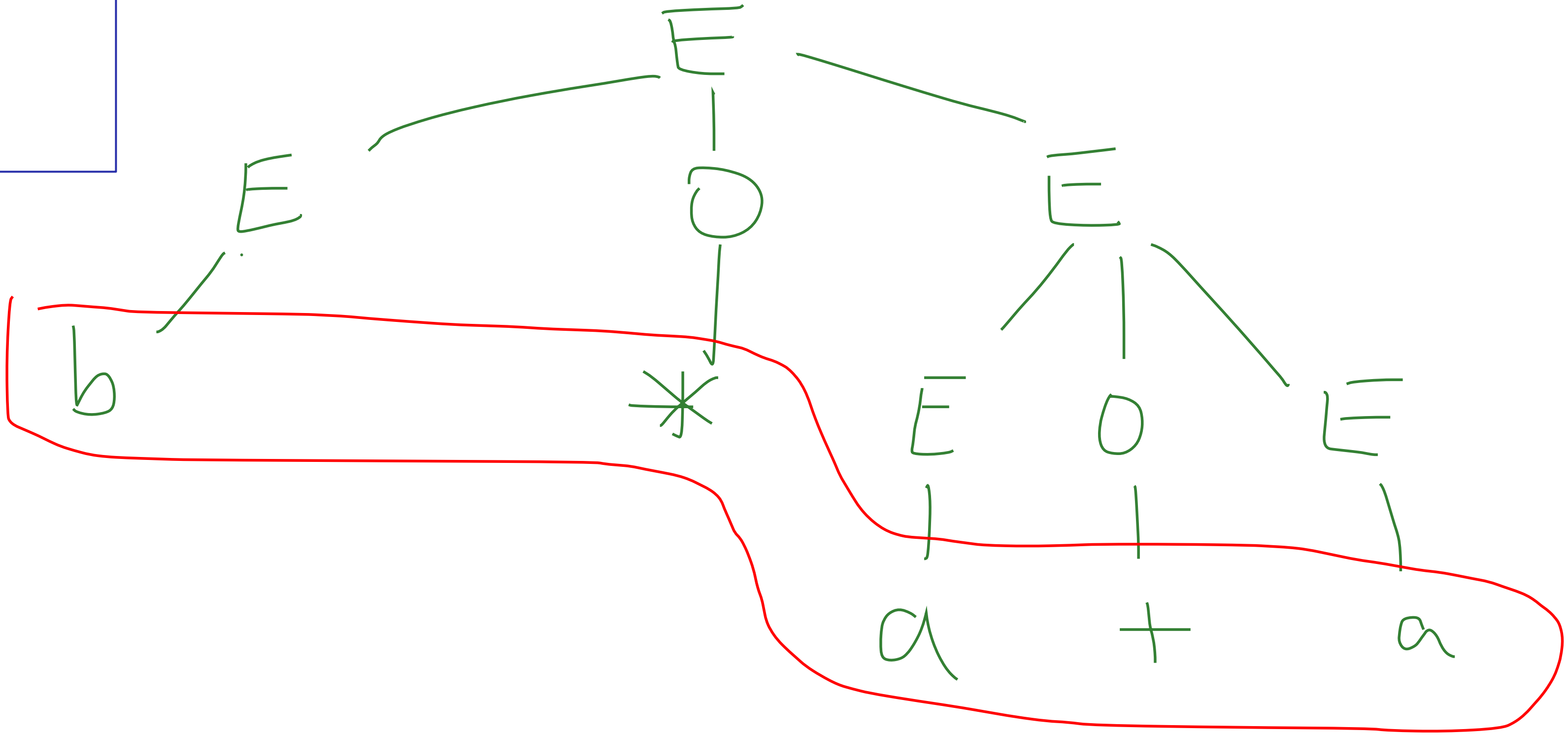
Parse trees show which rules were used when a string was derived

RULES of G_1:

E → a
E → b
E → EOE
O → +
O → *

Derivation of a string:

$$E \Rightarrow EOE \Rightarrow bOE \Rightarrow b{*}E \Rightarrow b{*}EOE \Rightarrow b{*}aOE \Rightarrow b{*}a{+}E \Rightarrow b{*}a{+}a$$



Parse trees show which rules were used when a string was derived

RULES of G_1:

$E \rightarrow a$

$E \rightarrow b$

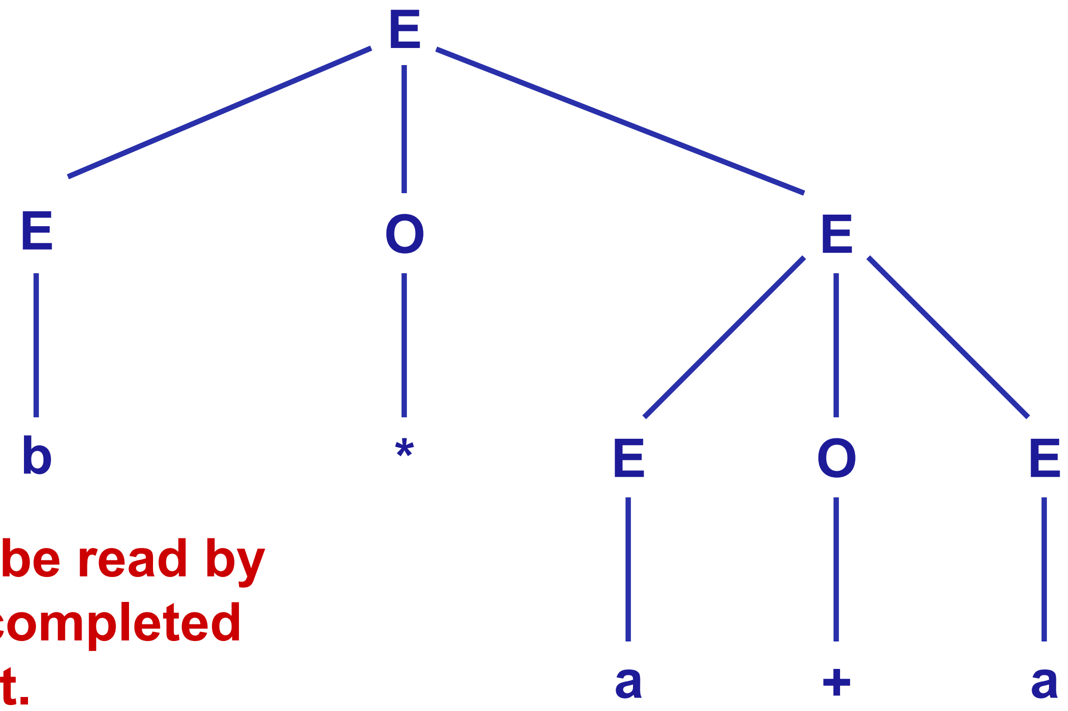
$E \rightarrow EOE$

$O \rightarrow +$

$O \rightarrow *$

Derivation of a string:

$E \Rightarrow EOE \Rightarrow bOE \Rightarrow b*E \Rightarrow b*EOE$
$\Rightarrow b*aOE \Rightarrow b*a+E \Rightarrow b*a+a$



The generated string may be read by reading the leaves of the completed parse tree from left to right.

Parse trees show which rules were used when a string was derived

RULES of G_1:

$E \rightarrow a$

$E \rightarrow b$

$E \rightarrow EOE$

$O \rightarrow +$

$O \rightarrow *$

Derivation of a string:

$E \Rightarrow EOE \Rightarrow bOE \Rightarrow b{*}E \Rightarrow b{*}EOE$
$\Rightarrow b{*}aOE \Rightarrow b{*}a{+}E \Rightarrow b{*}a{+}a$

***Second* derivation of same string:**

$E \Rightarrow EOE \Rightarrow E{*}E \Rightarrow E{*}EOE \Rightarrow E{*}EOa$
$\Rightarrow E{*}E{+}a \Rightarrow E{*}a{+}a \Rightarrow b{*}a{+}a$

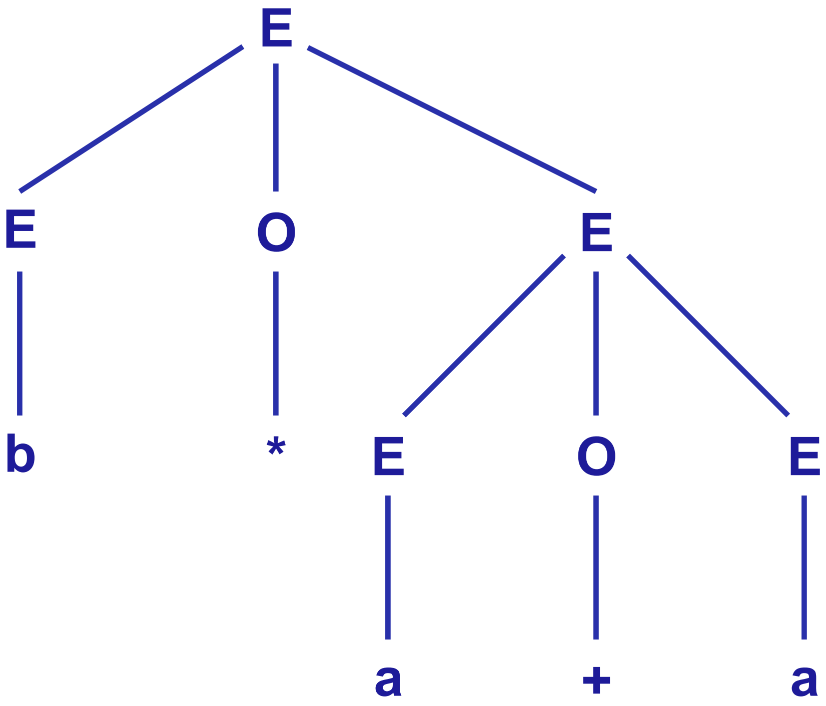
These 2 derivations of same string correspond to same parse tree

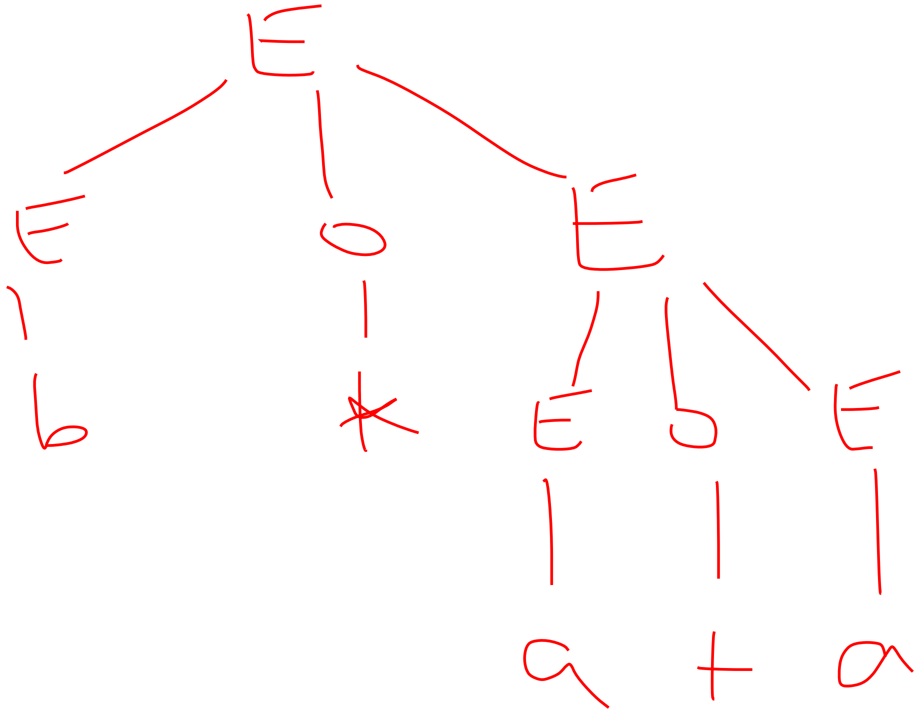
Derivation 1:

E ⇒ EOE ⇒ bOE ⇒ b*E
⇒ b*EOE ⇒ b*aOE
⇒ b*a+E ⇒ b*a+a

Derivation 2:

E ⇒ EOE ⇒ E*E ⇒ E*EOE
⇒ E*EOa ⇒ E*E+a
⇒ E*a+a ⇒ b*a+a





Parse trees show which rules were used when a string was derived

RULES of G_1:

$E \rightarrow a$
$E \rightarrow b$
$E \rightarrow EOE$
$O \rightarrow +$
$O \rightarrow *$

Derivation of a string:

$E \Rightarrow EOE \Rightarrow bOE \Rightarrow b{*}E \Rightarrow b{*}EOE$
$\Rightarrow b{*}aOE \Rightarrow b{*}a{+}E \Rightarrow b{*}a{+}a$

***Second* derivation of same string:**

$E \Rightarrow EOE \Rightarrow E{*}E \Rightarrow E{*}EOE \Rightarrow E{*}EOa$
$\Rightarrow E{*}E{+}a \Rightarrow E{*}a{+}a \Rightarrow b{*}a{+}a$

***Third* derivation of same string:**

$E \Rightarrow EOE \Rightarrow EOa \Rightarrow E{+}a \Rightarrow EOE{+}a$
$\Rightarrow EOa{+}a \Rightarrow E{*}a{+}a \Rightarrow b{*}a{+}a$

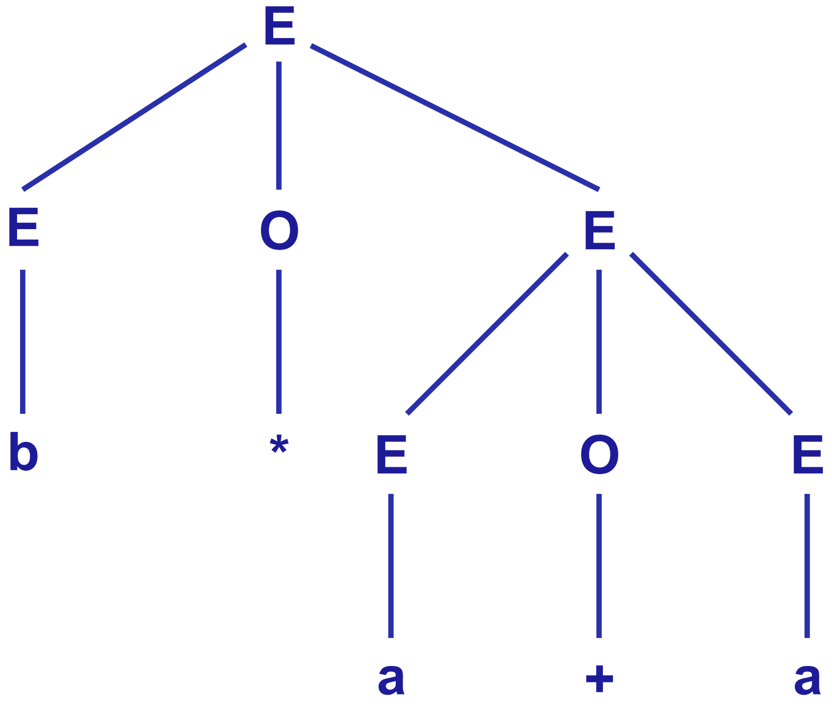
Derivations 1 and 3 correspond to *different* parse trees

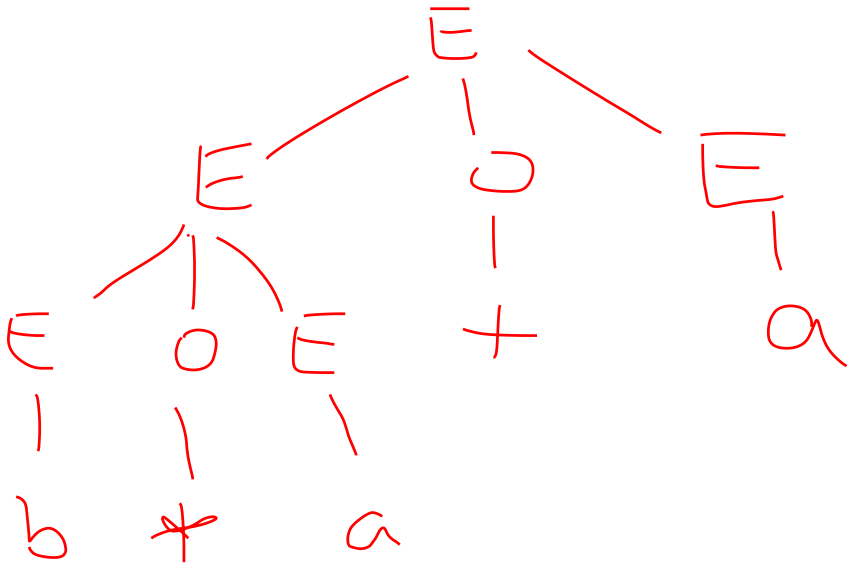
Derivation 1:

E ⇒ EOE ⇒ bOE ⇒ b*E
⇒ b*EOE ⇒ b*aOE
⇒ b*a+E ⇒ b*a+a

Derivation 3:

E ⇒ EOE ⇒ EOa ⇒ E+a
⇒ EOE+a ⇒ EOa+a
⇒ E*a+a ⇒ b*a+a





So, how should we interpret the string?

- Should b*a+a represent

  b*(a+a) or (b*a)+a ?

- The answer matters…consider value of the expression, say, when b=5 and a=2.
  - If compiler encounters this expression in source code, we don’t want any confusion about what the programmer intended

Ambiguity

- When a string can be derived from a grammar in two fundamentally different ways, we say the string can be *ambiguously derived.*
  - e.g., b*a+a is ambiguously derived in G_1

- When any string in a language is ambiguously derived in a grammar G, then G is said to be an *ambiguous grammar.*
  - e.g., G_1 is an ambiguous grammar because b+a+a is ambiguously derived in it

Ambiguity: Dangling Else Problem

- Suppose the CFG describing a programming language contains the following rules

  IfStmt → if B then S | if B then S else S

- How should this code be interpreted?

  if x>0 then if y<0 then output yes else output no

Ambiguity: Dangling Else Problem

IfStmt → if B then S | if B then S else S

if x>0 then if y<0 then output yes else output no

```
if x>0
then
        if y <0
        then output yes
        else output no
```

```
if x >0
then
        if y < 0
        then output yes
else output no
```

Ambiguity

- To demonstrate that a specific string s can be ambiguously derived, one can:
  - Give two different parse trees for s, or
  - Give two different *leftmost* derivations for s
    - (A leftmost derivation is one in which, at each step, the leftmost nonterminal remaining in the string is replaced.)
  - Note: Giving one leftmost and one rightmost derivation for s is *not* sufficient – these two derivations might correspond to the same parse tree.

Example: grammar G_5

111

- $A \rightarrow BC$
  $B \rightarrow 1B1 \mid 1$
  $C \rightarrow 1C1 \mid \varepsilon$

  - What is $L(G_5)$?
  - Show that grammar G_5 is ambiguous.

Leftmost derivation

$A \Rightarrow BC \Rightarrow 1C \Rightarrow 11C1 \Rightarrow 111$

$A \Rightarrow BC \Rightarrow 1B1C \Rightarrow 111C \Rightarrow 111$

Chomsky Normal Form (CNF)

- Chomsky Normal Form is a special format for CFGs, with restrictions on what the rules can look like
  - Named for Noam Chomsky, MIT linguist
  - Helpful in reasoning about what strings can be derived from a CFG

Chomsky Normal Form (CNF)

- Definition: A CFG G with start symbol S is in Chomsky Normal Form if every rule is in of one of the two following forms:

  $$A \rightarrow BC$$

  $$A \rightarrow a$$

  where a is any terminal, A is any variable, and B and C are any variables except S.

  In addition, the rule $S \rightarrow \varepsilon$ is allowed, but ε may not appear elsewhere in the grammar.

Example: a grammar in CNF

$S \rightarrow a \mid YZ \mid \varepsilon$

$Y \rightarrow ZZ \mid ZY \mid c$

$Z \rightarrow YY \mid b$

- For a CFG G in CNF, how many derivation steps are needed to generate a string s made up of n terminals?
  - Knowing this helps us design an algorithm to check if s is in L(G)

for string of length n,
need exactly 2n-1 steps

Theorem

- Any context-free language can be expressed by a context-free grammar in Chomsky Normal Form.
    - Proof: provide algorithm to convert any CFG into an equivalent CFG in CNF

Given a CFG, how can we put it into Chomsky Normal Form?

Step 1. Create a new start symbol S_0, and add rule $S_0 \rightarrow S$. This ensures that start symbol isn't on RHS of any rule.

Step 2. Remove rules of form $A \rightarrow \varepsilon$, and "fix up": for each rule with a RHS that includes A, add a copy of that rule with A removed. Do this for all combinations.

Step 3. Remove rules of form $A \rightarrow B$ (so-called "unit rules"), and "fix": for each rule B → *RHS* add a rule A → *RHS*

Step 4. Put remaining rules in proper form. May require introducing new variables. Reuse new variables where possible, to keep resulting grammar cleaner.

Practice

- Put the following CFG into CNF

$S \to ABS \mid \varepsilon$

$A \to xyz \mid \varepsilon$

$B \to wB \mid v$

In steps that follow, modifications are indicated in green

① Add new start symbol:

$\boxed{S_0 \to S}$

$S \to ABS \mid \varepsilon$

$A \to xyz \mid \varepsilon$

$B \to wB \mid v$

2a) Remove $A \to \varepsilon$

$S_0 \to S$

$S \to ABS \mid \varepsilon \mid \boxed{BS}$

$A \to xyz \mid \cancel{\varepsilon}$

$B \to wB \mid v$

2b) Remove $S \to \varepsilon$

$S_0 \to S \mid \boxed{\varepsilon}$

$S \to ABS \mid \cancel{\varepsilon} \mid BS \mid \boxed{AB \mid B}$

$A \to xyz$

$B \to wB \mid v$

3a Remove unit rule $S \rightarrow B$

$$S_0 \rightarrow S \mid \varepsilon$$

$$S \rightarrow ABS \mid BS \mid AB \mid \cancel{B} \mid \boxed{wB \mid v}$$

$$A \rightarrow xyz$$

$$B \rightarrow wB \mid v$$

3b Remove unit rule $S_0 \rightarrow S$

$$S_0 \rightarrow \cancel{S} \mid \varepsilon \mid \boxed{ABS \mid BS \mid AB \mid wB \mid v}$$

$$S \rightarrow ABS \mid BS \mid AB \mid wB \mid v$$

$$A \rightarrow xyz$$

$$B \rightarrow wB \mid v$$

(4) Put remaining rules in proper form

(a) Introduce new variable C:

$$S_0 \to \varepsilon \mid \boxed{AC} \mid BS \mid AB \mid wB \mid v$$

$$S \to \boxed{AC} \mid BS \mid AB \mid wB \mid v$$

$$A \to xyz$$

$$B \to wB \mid v$$

$$\boxed{C \to BS}$$

(b) Introduce new variable D:

$$S_0 \to \varepsilon \mid AC \mid BS \mid AB \mid \boxed{DB} \mid v$$

$$S \to AC \mid BS \mid AB \mid \boxed{DB} \mid v$$

$$A \to xyz$$

$$B \to \boxed{DB} \mid v$$

$$C \to BS$$

$$\boxed{D \to w}$$

(c) Introduce new variable E

$S_0 \to \varepsilon \mid AC \mid BS \mid AB \mid DB \mid v$

$S \to AC \mid BS \mid AB \mid DB \mid v$

$A \to \boxed{xE}$

$B \to DB \mid v$

$C \to BS$

$D \to w$

$\boxed{E \to yz}$

(d) Introduce new varibles F, G, H

$S_0 \to \varepsilon \mid AC \mid BS \mid AB \mid DB \mid v$

$S \to AC \mid BS \mid AB \mid DB \mid v$

$\boxed{A \to FE}$

$B \to DB \mid v$

$C \to BS$

$D \to w$

$\boxed{E \to GH}$

$\boxed{F \to x}$

$\boxed{\begin{array}{l} G \to y \\ \hline H \to z \end{array}}$