Chapter 12

Context-Free Grammars

12.1 Introduction

English grammar has rules for constructing sentences; e.g.,

1. A *sentence* can be a *subject* followed by a *predicate* .
2. A *subject* can be a *noun-phrase* .
3. A *noun-phrase* can be an *adjective* followed by a *noun-phrase* .
4. A *noun-phrase* can be an *article* followed by a *noun-phrase* .
5. A *noun-phrase* can be a *noun* .
6. A *predicate* can be a *verb* followed by a *noun-phrase* .
7. A *noun* can be:

   person *fish* *stapler* *book*

8. A *verb* can be:

   buries *touches* *grabs* *eats*

9. An *adjective* can be:

   big *small*

10. An *article* can be:

    the *a* *an*

These rules can be used to construct the following sentence:

The small person eats the big fish

$$
\begin{array}{lll}
\underline{\textit{sentence}} & \Rightarrow \underline{\textit{subject}}\ \underline{\textit{predicate}} & \text{Rule 1} \\
 & \Rightarrow \underline{\textit{noun-phrase}}\ \underline{\textit{predicate}} & \text{Rule 2} \\
 & \Rightarrow \underline{\textit{noun-phrase}}\ \underline{\textit{verb}}\ \underline{\textit{noun-phrase}} & \text{Rule 6} \\
 & \Rightarrow \underline{\textit{article}}\ \underline{\textit{noun-phrase}}\ \underline{\textit{verb}}\ \underline{\textit{noun-phrase}} & \text{Rule 4} \\
 & \Rightarrow \underline{\textit{article}}\ \underline{\textit{adjective}}\ \underline{\textit{noun-phrase}}\ \underline{\textit{verb}}\ \underline{\textit{noun-phrase}} & \text{Rule 3} \\
 & \Rightarrow \underline{\textit{article}}\ \underline{\textit{adjective}}\ \underline{\textit{noun}}\ \underline{\textit{verb}}\ \underline{\textit{noun-phrase}} & \text{Rule 5} \\
 & \Rightarrow \underline{\textit{article}}\ \underline{\textit{adjective}}\ \underline{\textit{noun}}\ \underline{\textit{verb}}\ \underline{\textit{article}}\ \underline{\textit{noun-phrase}} & \text{Rule 4} \\
 & \Rightarrow \underline{\textit{article}}\ \underline{\textit{adjective}}\ \underline{\textit{noun}}\ \underline{\textit{verb}}\ \underline{\textit{article}}\ \underline{\textit{adjective}}\ \underline{\textit{noun-phrase}} & \text{Rule 3} \\
 & \Rightarrow \underline{\textit{article}}\ \underline{\textit{adjective}}\ \underline{\textit{noun}}\ \underline{\textit{verb}}\ \underline{\textit{article}}\ \underline{\textit{adjective}}\ \underline{\textit{noun}} & \text{Rule 5} \\
 & \Rightarrow \text{the}\ \underline{\textit{adjective}}\ \underline{\textit{noun}}\ \underline{\textit{verb}}\ \underline{\textit{article}}\ \underline{\textit{adjective}}\ \underline{\textit{noun}} & \text{Rule 10} \\
 & \Rightarrow \text{the small}\ \underline{\textit{noun}}\ \underline{\textit{verb}}\ \underline{\textit{article}}\ \underline{\textit{adjective}}\ \underline{\textit{noun}} & \text{Rule 9} \\
 & \Rightarrow \text{the small person}\ \underline{\textit{verb}}\ \underline{\textit{article}}\ \underline{\textit{adjective}}\ \underline{\textit{noun}} & \text{Rule 7} \\
 & \Rightarrow \text{the small person eats}\ \underline{\textit{article}}\ \underline{\textit{adjective}}\ \underline{\textit{noun}} & \text{Rule 8} \\
 & \Rightarrow \text{the small person eats the}\ \underline{\textit{adjective}}\ \underline{\textit{noun}} & \text{Rule 10} \\
 & \Rightarrow \text{the small person eats the big}\ \underline{\textit{noun}} & \text{Rule 9} \\
 & \Rightarrow \text{the small person eats the big fish} & \text{Rule 7}
\end{array}
$$

Definition: The things that cannot be replaced by anything are called *terminals.*

Definition: The things that must be replaced by other things are called *nonterminals.*

In the above example,

- *small* and *eats* are terminals.
- *noun-phrase* and *verb* are nonterminals.

Example: restricted class of arithmetic expressions on integers.

$$
\begin{array}{lcl}
\underline{start} & \rightarrow & \underline{AE} \\
\underline{AE} & \rightarrow & \underline{AE} + \underline{AE} \\
\underline{AE} & \rightarrow & \underline{AE} - \underline{AE} \\
\underline{AE} & \rightarrow & \underline{AE} * \underline{AE} \\
\underline{AE} & \rightarrow & \underline{AE} \; / \; \underline{AE} \\
\underline{AE} & \rightarrow & \underline{AE} ** \underline{AE} \\
\underline{AE} & \rightarrow & (\underline{AE}\,) \\
\underline{AE} & \rightarrow & -\underline{AE} \\
\underline{AE} & \rightarrow & \underline{ANY\text{-}NUMBER}
\end{array}
$$

- nonterminals: $\underline{start}$, $\underline{AE}$
- terminals: $\underline{ANY\text{-}NUMBER}$, +, −, ∗, /, ∗∗, (,)
- Can generate the arithmetic expression

  $\underline{ANY\text{-}NUMBER} + (\underline{ANY\text{-}NUMBER} - \underline{ANY\text{-}NUMBER}\,) / \underline{ANY\text{-}NUMBER}$

  as follows:

$$
\begin{array}{ll}
\underline{start} & \Rightarrow \underline{AE} \\
& \Rightarrow \underline{AE} + \underline{AE} \\
& \Rightarrow \underline{AE} + \underline{AE} \; / \; \underline{AE} \\
& \Rightarrow \underline{AE} + (\underline{AE}\,) \; / \; \underline{AE} \\
& \Rightarrow \underline{AE} + (\underline{AE} - \underline{AE}\,) \; / \; \underline{AE} \\
& \Rightarrow \underline{ANY\text{-}NUMBER} + (\underline{AE} - \underline{AE}\,) \; / \; \underline{AE} \\
& \Rightarrow \underline{ANY\text{-}NUMBER} + (\underline{ANY\text{-}NUMBER} - \underline{AE}\,) \; / \; \underline{AE} \\
& \Rightarrow \underline{ANY\text{-}NUMBER} + (\underline{ANY\text{-}NUMBER} - \underline{ANY\text{-}NUMBER}\,) \; / \; \underline{AE} \\
& \Rightarrow \underline{ANY\text{-}NUMBER} + (\underline{ANY\text{-}NUMBER} - \underline{ANY\text{-}NUMBER}\,) \; / \; \underline{ANY\text{-}NUMBER}
\end{array}
$$

Could also make *ANY-NUMBER* a nonterminal:

Rule 1 *ANY-NUMBER* → *FIRST-DIGIT*
Rule 2 *FIRST-DIGIT* → *FIRST-DIGIT* *OTHER-DIGIT*
Rule 3 *FIRST-DIGIT* → 1 2 3 4 5 6 7 8 9
Rule 4 *OTHER-DIGIT* → 0 1 2 3 4 5 6 7 8 9

In this case,

- nonterminals: *ANY-NUMBER* , *FIRST-DIGIT* , *OTHER-DIGIT*
- terminals: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Can produce the number 90210 as follows:

Rule 1 *ANY-NUMBER* ⇒ *FIRST-DIGIT*
Rule 2 ⇒ *FIRST-DIGIT* *OTHER-DIGIT*
Rule 2 ⇒ *FIRST-DIGIT* *OTHER-DIGIT* *OTHER-DIGIT*
Rule 2 ⇒ *FIRST-DIGIT* *OTHER-DIGIT* *OTHER-DIGIT* *OTHER-DIGIT*
Rule 2 ⇒ *FIRST-DIGIT* *OTHER-DIGIT* *OTHER-DIGIT* *OTHER-DIGIT* *OTHER-DIGIT*
Rule 3 ⇒ 9 *OTHER-DIGIT* *OTHER-DIGIT* *OTHER-DIGIT* *OTHER-DIGIT*
Rule 4 ⇒ 9 0 *OTHER-DIGIT* *OTHER-DIGIT* *OTHER-DIGIT*
Rule 4 ⇒ 9 0 2 *OTHER-DIGIT* *OTHER-DIGIT*
Rule 4 ⇒ 9 0 2 1 *OTHER-DIGIT*
Rule 4 ⇒ 9 0 2 1 0

Note that we had rules of the form:

one nonterminal → string of nonterminals

or

one nonterminal → choice of terminals

Definition: The sequence of applications of the rules that produces the finished string of terminals from the starting symbol is called a *derivation* or *production.*

12.2 Context-Free Grammars

Example: terminals: $\Sigma = \{a\}$
nonterminal: $\Omega = \{S\}$
productions:

$$\begin{aligned} S &\rightarrow aS \\ S &\rightarrow \Lambda \end{aligned}$$

- Can generate a^4 as follows:

$$\begin{aligned} S &\Rightarrow aS \\ &\Rightarrow aaS \\ &\Rightarrow aaaS \\ &\Rightarrow aaaaS \\ &\Rightarrow aaaa\Lambda = aaaa \end{aligned}$$

Example: terminal: a
nonterminal: S
productions:

$$\begin{aligned} S &\rightarrow SS \\ S &\rightarrow a \\ S &\rightarrow \Lambda \end{aligned}$$

Can write this in more compact notation:

$$S \rightarrow SS \mid a \mid \Lambda$$

which is called the *Backus Normal Form* or *Backus-Naur Form* (BNF).

- CFL is $\mathbf{a}^*$
- Can generate a^2 as follows:

$$\begin{aligned} S &\Rightarrow SS \\ &\Rightarrow SSS \\ &\Rightarrow SSa \\ &\Rightarrow SSSa \\ &\Rightarrow SaSa \\ &\Rightarrow \Lambda aSa \\ &\Rightarrow \Lambda a\Lambda a = aa \end{aligned}$$

- In previous example, unique way to generate any word.
- Here, each word in CFL has infinitely many derivations.

Definition: A *context-free grammar* (CFG) is a collection $G = (\Sigma, \Omega, R, S)$, with

1. A (finite) alphabet Σ of letters called *terminals* from which we make strings that will be the words of the language.
2. A finite set Ω of symbols called *nonterminals*, one of which is the symbol S (i.e., $S \in \Omega$), standing for "start here."
3. A finite set R of *productions*, with $R \subset \Omega \times (\Sigma + \Omega)^*$. If a production $(N, \mathcal{U}) \in R$ with $N \in \Omega$ and $\mathcal{U} \in (\Sigma + \Omega)^*$, then we write the production as
$$N \rightarrow \mathcal{U}.$$
Thus, each production is of the form

   one nonterminal $\rightarrow$ finite string of terminals and/or nonterminals

   where the strings of terminals, nonterminals can consist of only terminals or of only nonterminals, or any mixture of terminals and nonterminals or even the empty string. We require that at least one production has the nonterminal S as its left side.

Convention:

- Terminals will typically be smallcase letters.
- Nonterminals will typically be uppercase letters.

Definition: The *language generated* (defined, derived, produced) by a CFG G is the set of all strings of terminals that can be produced from the start symbol S using the productions as substitutions. A language generated by a CFG G is called a *context-free language* (CFL) and is denoted by $L(G)$.

Example: terminals: $\Sigma = \{a\}$
nonterminal: $\Omega = \{S\}$
productions:

$$\begin{aligned} S &\rightarrow aS \\ S &\rightarrow \Lambda \end{aligned}$$

- Let L_1 the language generated by this CFG, and let L_2 be the language generated by regular expression $\mathbf{a}^*$.
- **Claim**: $L_1 = L_2$.
- **Proof:**
  - * We first show that $L_2 \subset L_1$.
    - · Consider $a^n \in L_2$ for $n \geq 1$. We can generate a^n by using first production n times, and then second production.
    - · Can generate $\Lambda \in L_2$ by using second production only.
    - · Hence $L_2 \subset L_1$.
  - * We now show that $L_1 \subset L_2$.
    - · Since a is the only terminal, CFG can only produce strings having only a's.
    - · Thus, $L_1 \subset L_2$.

Note that

- Two types of arrows:
  $\rightarrow$ used in statement of productions
  $\Rightarrow$ used in derivation of word
- in the above derivation of a^4, there were many unfinished stages that consisted of both terminals and nonterminals. These are called *working strings*.
- Λ is neither a nonterminal (since it cannot be replaced with something else) nor a terminal (since it disappears from the string).

12.3 Examples

Example: terminals: a, b
nonterminals: S
productions:

$$\begin{aligned} S &\rightarrow aS \\ S &\rightarrow bS \\ S &\rightarrow a \\ S &\rightarrow b \end{aligned}$$

More compact notation:

$$S \to aS \mid bS \mid a \mid b$$

- Can produce the word *abbab* as follows:

$$\begin{aligned} S &\Rightarrow aS \\ &\Rightarrow abS \\ &\Rightarrow abbS \\ &\Rightarrow abbaS \\ &\Rightarrow abbab \end{aligned}$$

- Let L_1 be the CFL, and let L_2 be the language generated by the regular expression $(\mathbf{a} + \mathbf{b})^+$.
- **Claim:** $L_1 = L_2$.
- **Proof:**
- First we show that $L_2 \subset L_1$.
  * Consider any string $w \in L_2$.
  * Read letters of w from left to right.
  * For each letter read in, if it is not the last, then
    · use the production $S \to aS$ if the letter is a or
    · use the production $S \to bS$ if the letter is b
  * For the last letter of the word,
    · use the production $S \to a$ if the letter is a or
    · use the production $S \to b$ if the letter is b
  * In each stage of the derivation, the working string has the form

$$\text{(string of terminals)}S$$

- Hence, we have shown how to generate w using the CFG, which means that $w \in L_1$.
- Hence, $L_2 \subset L_1$.

• Now we show that $L_1 \subset L_2$.

- To show this, we need to show that if $w \in L_1$, then $w \in L_2$.
- This is equivalent to showing that if $w \notin L_2$, then $w \notin L_1$.

- Note that the only string $w \notin L_2$ is $w = \Lambda$.
- But note that Λ cannot be generated by the CFG, so $\Lambda \notin L_1$.
- Hence, we have proven that $L_1 \subset L_2$.

Example: terminals: a, b
nonterminals: S, X, Y
productions:

$$\begin{aligned} S &\rightarrow X \mid Y \\ X &\rightarrow \Lambda \\ Y &\rightarrow aY \mid bY \mid a \mid b \end{aligned}$$

- Note that if we use first production ($S \rightarrow X$), then the only word we can generate is Λ.
- The second production ($S \rightarrow Y$) leads to a collection of productions identical to the previous example.
- Thus, the second production produces $(\mathbf{a}+\mathbf{b})^+$.
- CFL is $(\mathbf{a}+\mathbf{b})^*$

Example: terminals: a, b
nonterminals: S
productions:

$$S \rightarrow aS \mid bS \mid a \mid b \mid \Lambda$$

- CFL is $(\mathbf{a}+\mathbf{b})^*$
- For this CFG, the sequence of productions to generate any word is not unique.
- e.g., can generate bab using

$$\begin{aligned} S &\Rightarrow bS \\ &\Rightarrow baS \\ &\Rightarrow babS \\ &\Rightarrow bab\Lambda = bab \end{aligned}$$

or

$$\begin{aligned} S &\Rightarrow bS \\ &\Rightarrow baS \\ &\Rightarrow bab \end{aligned}$$

Example: terminals: a, b
nonterminals: S, X
productions:

$$\begin{aligned} S &\rightarrow XaaX \\ X &\rightarrow aX \mid bX \mid \Lambda \end{aligned}$$

- The last set of productions generates any word from Σ^*.
- CFL is $(\mathbf{a}+\mathbf{b})^*\mathbf{aa}(\mathbf{a}+\mathbf{b})^*$
- Can generate *abbaaba* as follows:

$$\begin{aligned} S &\Rightarrow XaaX \\ &\Rightarrow aXaaX \\ &\Rightarrow abXaaX \\ &\Rightarrow abbXaaX \\ &\Rightarrow abb\Lambda aaX = abbaaX \\ &\Rightarrow abbaabX \\ &\Rightarrow abbaabaX \\ &\Rightarrow abbaaba\Lambda = abbaaba \end{aligned}$$

Example: terminals: a, b
nonterminals: S, X, Y
productions:

$$\begin{aligned} S &\rightarrow XY \\ X &\rightarrow aX \mid bX \mid a \\ Y &\rightarrow Ya \mid Yb \mid a \end{aligned}$$

- X productions can produce words ending with a.

- Y productions can produce words starting with a.
- CFL is $(\mathbf{a}+\mathbf{b})^*\mathbf{aa}(\mathbf{a}+\mathbf{b})^*$
- Can generate $abbaaba$ as follows:

$$\begin{aligned} S &\Rightarrow XY \\ &\Rightarrow aXY \\ &\Rightarrow abXY \\ &\Rightarrow abbXY \\ &\Rightarrow abbaY \\ &\Rightarrow abbaYa \\ &\Rightarrow abbaYba \\ &\Rightarrow abbaaba \end{aligned}$$

Example: Give CFGs for each of the following languages over the alphabet $\Sigma = \{a, b\}$:

1. $\{a^n b^n : n \geq 0\}$
2. PALINDROME
3. EVEN-PALINDROME
4. ODD-PALINDROME

Example: terminals: a, b
nonterminals: S, B, U
productions:

$$\begin{aligned} S &\rightarrow SS \mid BS \mid SB \mid \Lambda \mid USU \\ B &\rightarrow aa \mid bb \\ U &\rightarrow ab \mid ba \end{aligned}$$

Show that this generates EVEN-EVEN

- Note that starting from B, we can generate a balanced pair, i.e., either aa or bb.

- Starting from U, we can generate an unbalanced pair, i.e., either ab or ba.
- First show that every word in EVEN-EVEN can be generated using these productions.
  - Recall that EVEN-EVEN has regular expression

    $$[\mathbf{aa} + \mathbf{bb} + (\mathbf{ab} + \mathbf{ba})(\mathbf{aa} + \mathbf{bb})^*(\mathbf{ab} + \mathbf{ba})]^*$$

  - Three types of syllables:
    1. $\mathbf{aa}$,
    2. $\mathbf{bb}$,
    3. $(\mathbf{ab} + \mathbf{ba})(\mathbf{aa} + \mathbf{bb})^*(\mathbf{ab} + \mathbf{ba})$
  - Consider any word generated from the regular expression for EVEN-EVEN. Let's examine the way it was generated using the regular expression, and show how to generate the same word using our CFG.
  - Start our derivation using the CFG from S.
  - Every time we iterate the outer star in the regular expression, we choose one of the three syllables.
    1. If we choose a syllable of type 1, then first use the production $S \to BS$ and then the production $B \to aa$. Thus, we end up with a working string of aaS for this iteration of the outer star.
    2. If we choose a syllable of type 2, then first use the production $S \to BS$ and then the production $B \to bb$. Thus, we end up with a working string of bbS for this iteration of the outer star.
    3. If we choose a syllable of type 3, then
       (a) First use the production $S \to SS$.
       (b) Then change the first S using the production $S \to USU$, resulting in $USUS$.
       (c) If the first $(\mathbf{ab}+\mathbf{ba})$ in the syllable $(\mathbf{ab}+\mathbf{ba})(\mathbf{aa}+\mathbf{bb})^*(\mathbf{ab}+\mathbf{ba})$ is used to generate ab, then replace the first U in $USUS$ using the production $U \to ab$, resulting in $abSUS$. If the first $(\mathbf{ab} + \mathbf{ba})$ in $(\mathbf{ab} + \mathbf{ba})(\mathbf{aa} + \mathbf{bb})^*(\mathbf{ab} + \mathbf{ba})$ is used to generate ba, then replace the first U in $USUS$ using the production $U \to ba$, resulting in $baSUS$. Do the same for the second $(\mathbf{ab}+\mathbf{ba})$ in $(\mathbf{ab}+\mathbf{ba})(\mathbf{aa}+\mathbf{bb})^*(\mathbf{ab}+\mathbf{ba})$. Thus,

we now have $xSyS$ as a working string for this iteration of the outer star of the regular expression, where x is either ab or ba, and y is either ab or ba.

(d) Now suppose the $(\mathbf{aa} + \mathbf{bb})^*$ is iterated n times, $n \geq 0$. If $n = 0$, then change the first S in $xSyS$ using the production $S \to \Lambda$, resulting in $x\Lambda yS = xyS$. If $n \geq 1$, then change the first S in $xSyS$ using the production $S \to BS$ and do this n times, resulting in $xBBB\cdots BSyS$, where there are n B's in the clump of B's. Then change the first S using the production $S \to \Lambda$, resulting in $xBBB\cdots B\Lambda yS = xBBB\cdots ByS$, where there are n B's in the clump of B's. Finally, if on the kth iteration, $k \leq n$, of the $*$ in $(\mathbf{aa} + \mathbf{bb})^*$ we generated aa, then replace the kth B using the production $B \to aa$. If on the kth iteration, $k \leq n$, of the $*$ in $(\mathbf{aa} + \mathbf{bb})^*$ we generated bb, then replace the kth B using the production $B \to bb$.

- After completing all of the iterations of the outer star in the regular expression, use the production $S \to \Lambda$.
- e.g., for word $babbabaa \in$ EVEN-EVEN,

$$
\begin{aligned}
S &\Rightarrow SS \\
&\Rightarrow USUS \\
&\Rightarrow baSUS \\
&\Rightarrow baBSUS \\
&\Rightarrow babbSUS \\
&\Rightarrow babb\Lambda US = babbUS \\
&\Rightarrow babbabS \\
&\Rightarrow babbabBS \\
&\Rightarrow babbabaaS \\
&\Rightarrow babbabaa\Lambda = babbabaa
\end{aligned}
$$

- Now show that all words generated by these productions are in EVEN-EVEN.
  - all words derived from S can be decomposed into two-letter syllables.
  - unbalanced syllables (ab and ba) come into working string in pairs, which adds two a's and two b's.

  - balanced syllables add two of one letter and none of the other
  - thus, the sum total of a's will be even, and the sum total of b's will be even
  - Thus, word generated by productions will be in EVEN-EVEN.

Example: terminals: a, b
nonterminals: S, A, B
productions:

$$\begin{aligned} S &\rightarrow aB \mid bA \\ A &\rightarrow a \mid aS \mid bAA \\ B &\rightarrow b \mid bS \mid aBB \end{aligned}$$

This generates the language EQUAL, which consists of all strings of positive length and that have an equal number of a's and b's.

Proof. Need to show two things:

1. every word in EQUAL can be generated using our productions.
2. every word generated by our productions is in EQUAL.

First we show 1.

- We make three claims:

  Claim 1: All words in EQUAL can be generated by some sequence of productions beginning with the start symbol S.

  Claim 2: All words that have one more a than b's can be generated from these productions by starting with the nonterminal A.

  Claim 3: All words that have one more b than a's can be generated from these productions by starting with the nonterminal B.

- We will prove that these three claims hold by contradiction.
- Assume that one of the three claims does not hold.
- Then there is some smallest word w that violates one of the claims.
- All words shorter than w must satisfy the three claims.

- First assume that w violates Claim 1.
  - This means that w is in EQUAL but cannot be generated starting with S.
  - Assume that w starts with a and that $w = aw_1$.
  - Since $w \in$ EQUAL, w_1 must have exactly one more b than a's.
  - However, w_1 is shorter than w.
  - Thus, we must be able to generate w_1 starting with B; i.e.,

    $$B \Rightarrow \cdots \Rightarrow w_1$$

  - But then

    $$S \Rightarrow aB \Rightarrow \cdots \Rightarrow aw_1 = w$$

    which is a contradiction.
  - We similarly reach a contradiction when the first letter of w is b.
  - Thus, w cannot violate Claim 1.
- Now assume that w violates Claim 2.
  - This means that w has one more a than b's but cannot be generated starting with A.
  - First assume that w starts with a.
    - Then $w = aw_1$, where $w_1 \in$ EQUAL.
    - Since w_1 is shorter than w, we must be able to generate w_1 starting with S; i.e.,

      $$S \Rightarrow \cdots \Rightarrow w_1$$

    - But then

      $$A \Rightarrow aS \Rightarrow \cdots \Rightarrow aw_1 = w$$

      which is a contradiction.
  - Now assume that w starts with b.
    - Then if we write $w = bw_1$, then w_1 has two more a's than b's.
    - We now split $w_1 = w_{11}w_{12}$, where w_{11} is the part of w_1 scanning from left to right until there is exactly one more a than b's, and let w_{12} be the rest of w_1.
    - Note that w_{12} also has exactly one more a than b's.

  * Since w_{11} and w_{12} are both shorter than w, we must be able to generate each of them starting with A; i.e.,

    $$A \Rightarrow \cdots \Rightarrow w_{11}$$

    and

    $$A \Rightarrow \cdots \Rightarrow w_{12}$$

  * But then

    $$A \Rightarrow bAA \Rightarrow \cdots \Rightarrow bw_{11}w_{12} = bw_1 = w$$

    which is a contradiction.

  - Thus we have shown that Claim 2 must hold.

- We can similarly show that Claim 3 must hold.
- Thus, all 3 claims hold, and so in particular, Claim 1 holds: all words in EQUAL can be generated starting from S.

Now we show 2 holds: every word generated by our productions is in EQUAL.

- We again make 3 claims

  Claim 4 All words generated from S are in EQUAL.

  Claim 5 All words generated from A have one more a than b's.

  Claim 6 All words generated from B have one more b than a's.

- We will show that these 3 claims hold by contradiction.
- Assume that one of the three claims does not hold.
- Then there is some smallest word w generated from S, A, or B that does not have the required property.
- All words shorter than w must satisfy the three claims.
- First assume that w violates Claim 4.
  - We have assumed that w can be generated from S but is not in EQUAL.
  - Assume that the first letter of w is a.

  - Then w was generated by first using the production $S \rightarrow aB$.
  - To generate w, this B generates a word w_1 which is shorter than w and by assumption w_1 has one more b than a's.
  - This implies that w has an equal number of a's and b's, which is a contradiction.
  - We get a similar contridiction if the first letter of w is b.

- Now assume that w violates Claim 5.
  - We have assumed that w can be generated from A but does not have exactly one more a than b's.
  - w could not have been generated by $A \rightarrow a$ since $w = a$, which satisfies the requirement.
  - Suppose w was generated by first using the production $A \rightarrow aS$.
    * Then to generate the rest of w, we would have to start from S to generate w_1, where $w = aw_1$.
    * However, since w_1 is shorter than w and w_1 is generated starting with S, we must have that $w_1 \in$ EQUAL.
    * This implies that w has exactly one more a than b's, which is a contradiction.
  - Suppose w was generated by first using the production $A \rightarrow bAA$.
    * To generate the rest of w, each of the A's need to generate strings w_1 and w_2 which are shorter than w such that $w = bw_1w_2$.
    * However, since w_1 and w_2 are shorter than w, we must have that w_1 and w_2 each have exactly one more a than b's.
    * Hence, $w = bw_1w_2$ must have exactly one more a than b's, which is a contradiction.
  - Thus, we have shown that Claim 5 must hold

- We can similarly show that Claim 6 must hold.

- Thus, all of the claims hold, and in particular, Claim 4: all words generated from $S \in$ EQUAL.

12.4 Trees

Can use a tree to illustrate how a word is derived from a CFG.

Definition: These trees are called *syntax trees*, *parse trees*, *generation trees*, *production trees*, or *derivation trees*.

Example: CFG:
terminals: a, b
nonterminals: S, A
productions:

$$\begin{aligned} S &\rightarrow AAA \mid A \\ A &\rightarrow AA \mid aA \mid Ab \mid a \mid b \end{aligned}$$

String $abaaba$ has the following derivation:

$$\begin{aligned} S &\Rightarrow AAA \\ &\Rightarrow aAAA \\ &\Rightarrow abAA \\ &\Rightarrow abAbA \\ &\Rightarrow abaAbA \\ &\Rightarrow abaabA \\ &\Rightarrow abaaba \end{aligned}$$

which corresponds to the following derivation tree:

```
          S
        / | \
       /  |  \
      /   |   \
     /    |    \
    A     A     A
  / |     | \   |
 /  |     |  \  |
a   A     A   b a
    |    / \
```

```
     |   /   \
     b  a     A
              |
              |
              a
```

Example: CFG for simplified arithmetic expressions.
terminals: $+$, $*$, $0, 1, 2, \ldots, 9$
nonterminals: S
productions:

$$S \to S + S \mid S * S \mid 0 \mid 1 \mid 2 \mid \cdots \mid 9$$

- Consider the expression $2 * 3 + 4$.
- Ambiguous how to evaluate this:
- Does this mean $(2 * 3) + 4 = 10$ or $2 * (3 + 4) = 14$?
- Can eliminate ambiguity by examining the two possible derivation trees

```
                S                        S
              / | \                    / | \
             /  |  \                  /  |  \
            /   |   \                /   |   \
           /    |    \              /    |    \
          S     +     S            S     *     S
        / | \         |            |         / | \
       /  |  \        |            |        /  |  \
      2   *   3       4            2       3   +   4
```

Eliminate the S's as follows:

```
           +                        *
         /   \                    /   \
        /     \                  /     \
       /       \                /       \
      /         \              /         \
     *           4            2           +
   /   \                                /   \
  /     \                              /     \
 2       3                             3       4
```

Note that we can construct a new notation for mathematical expressions:

- start at top of tree
- walk around tree keeping left hand touching tree
- first time hit each terminal, print it out.

This gives us a string which is in *operator prefix notation* or *Polish notation.*

In above examples,

- first tree yields
$$+ \ * \ 2\ 3\ 4$$
- second tree yields
$$* \ 2 \ + \ 3\ 4$$

To evaluate the string:

1. scan string from left to right.
2. the first time we read a substring of the form "operator-operand-operand" (o-o-o), replace the three symbols with the one result of the indicated arithmetic calculation.
3. go back to step 1

Example: (from above)

- first tree yields:

string	first o-o-o substring
$+ * 2\ 3\ 4$	$* 2\ 3$
$+ 6\ 4$	$+ 6\ 4$
10	

- second tree yields:

string	first o-o-o substring
$* 2 + 3\ 4$	$+ 3\ 4$
$* 2\ 7$	$* 2\ 7$
14	

Example: Consider the arithmetic expression:

$$3 + 4 * 6 + 2 + 8 + 1 * 5 + 9 * 7$$

There are many ways to evaluate this expression, one of which is as

$$((3 + 4) * (6 + 2) + ((8 + 1) * 5) + 9) * 7$$

This interpretation has

- derivation tree:

```
                *
               / \
              +   7
             / \
            +   9
           / \
          /   \
         /     \
        *       *
       / \     / \
      /   \   /   \
     +     +  +    5
    / \   / \/ \
    3 4   6 2 8 1
```

- prefix notation:

$$* \ + \ + \ * \ + 3\ 4 \ + \ 6\ 2 \ * \ + 8\ 1\ 5\ 9\ 7$$

- can evaluate prefix notation expression:

string	first o-o-o substring
$* + + * + 3\ 4 + 6\ 2 * + 8\ 1\ 5\ 9\ 7$	$+\ 3\ 4$
$* + + * 7 + 6\ 2 * + 8\ 1\ 5\ 9\ 7$	$+\ 6\ 2$
$* + + * 7\ 8 * + 8\ 1\ 5\ 9\ 7$	$*\ 7\ 8$
$* + + 56 * + 8\ 1\ 5\ 9\ 7$	$+\ 8\ 1$
$* + + 56 * 9\ 5\ 9\ 7$	$*\ 9\ 5$
$* + + 56\ 45\ 9\ 7$	$+\ 56\ 45$
$* + 101\ 9\ 7$	$+\ 101\ 9$
$* 110\ 7$	$*\ 110\ 7$
770	

Example:
terminals: a, b
nonterminals: S, A, B
productions:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

Can produce word ab in two ways:

1. $S \Rightarrow AB \Rightarrow aB \Rightarrow ab$
2. $S \Rightarrow AB \Rightarrow Ab \Rightarrow ab$

However, both derivations have the same syntax tree:

```
   S
  / \
 A   B
 |   |
 a   b
```

Definition: A CFG is *ambiguous* if for at least one word in its CFL there are two possible derivations of the word that correspond to two different syntax trees.

Example: PALINDROME
terminals: a, b
nonterminals: S
productions:

$$S \to aSa \mid bSb \mid a \mid b \mid \Lambda$$

Can generate the word *babbab* as follows:

$$\begin{aligned} S &\Rightarrow bSb \\ &\Rightarrow baSab \\ &\Rightarrow babSbab \\ &\Rightarrow babbab \end{aligned}$$

which has derivation tree:

```
  S
 /|\
b S b
 /|\
a S a
 /|\
b S b
  |
  ^
```

Can show that this CFG is *unambiguous.*

Example:
terminals: a, b
nonterminals: S
productions:

$$S \to aS \mid Sa \mid a$$

The word aa can be generated by two different trees:

```
    S          S
   / \        / \
  a   S      S   a
      |      |
      a      a
```

Therefore, this CFG is ambiguous.

Example: terminals: a, b
nonterminals: S
productions:

$$S \to aS \mid a$$

The CFL for this CFG is the same as above.

The word aa can now be generated by only one tree:

```
    S
   / \
  a   S
      |
      a
```

Therefore, this CFG is unambiguous.

Example:
terminals: a, b
nonterminals: S, X
productions:

$$\begin{array}{rcl} S & \rightarrow & aS \mid aSb \mid X \\ X & \rightarrow & Xa \mid a \end{array}$$

The word aa has two different derivations that correspond to different syntax trees:

1. $S \Rightarrow aS \Rightarrow aX \rightarrow aa$

```
   S
  / \
 a   S
     |
     X
     |
     a
```

2. $S \Rightarrow X \Rightarrow Xa \rightarrow aa$

```
   S
   |
   X
  / \
 X   a
 |
 a
```

Thus, this CFG is ambiguous.

Definition: For a given CFG, the *total language tree* is the tree

- with root S,
- whose children are all the productions of S,
- whose second descendents are all the working strings that can be constructed by applying one production to the leftmost nonterminal in each of the children,
- and so on.

Example:
terminals: a, b
nonterminals: S, X
productions:

$$\begin{aligned} S &\rightarrow aX \mid Xa \mid aXbXa \\ X &\rightarrow ba \mid ab \end{aligned}$$

This CFG has total language tree as follows:

```
             S
           / | \
          /  |  \
         /   |   \
        /    |    \
       /     |     \
     aX     Xa      aXbXa
    / |    / |      /    \
   /  |   /  |     /      \
aba aab baa aba ababXa     aabbXa
                /    \      /   \
          ababbaa abababa aabbbaa aabbaba
```

The CFL is finite.

Example:
terminals: a, b
nonterminals: S, X
productions:

$$
\begin{aligned}
S &\rightarrow aSb \mid aX \\
X &\rightarrow bX \mid a
\end{aligned}
$$

Total language tree:

```
                       S
                     /   \
                    /     \
                   /       \
                  /         \
                 /           \
                /             \
             aSb               aX
            /   \             /   \
           /     \           /     \
       aaSbb     aaXb       abX     aa
      /    \    /   \      /   \
aaaSbbb aaaXbb aabXb aaab abbX aba
    .      .    .           .
    .      .    .           .
    .      .    .           .
```

CFL is infinite.

Example: terminals: a
nonterminals: S, X
productions:

$$\begin{aligned} S &\rightarrow X \mid a \\ X &\rightarrow aX \end{aligned}$$

Total language tree:

```
     S
   /   \
  X     a
  |
 aX
  |
 aaX
  .
  .
  .
```

Tree is infinite, but CFL $= \{a\}$.