Chapter Ten: Grammars
Grammar is another of those common words for which the study of formal language introduces a precise technical definition. For us, a grammar is a certain kind of collection of rules for building strings. Like DFAs, NFAs, and regular expressions, grammars are mechanisms for defining languages rigorously.

A simple restriction on the form of these grammars yields the special class of right-linear grammars. The languages that can be defined by right-linear grammars are exactly the regular languages. There it is again!
Outline

• 10.1 A Grammar Example for English
• 10.2 The 4-Tuple
• 10.3 The Language Generated by a Grammar
• 10.4 Every Regular Language Has a Grammar
• 10.5 Right-Linear Grammars
• 10.6 Every Right-linear Grammar Generates a Regular Language
A Little English

• An article can be the word *a* or *the*:
  
  \[
  A \rightarrow a \\
  A \rightarrow the
  \]

• A noun can be the word *dog*, *cat* or *rat*:
  
  \[
  N \rightarrow dog \\
  N \rightarrow cat \\
  N \rightarrow rat
  \]
  
  *A noun phrase is an article followed by a noun:*
  
  \[
  P \rightarrow AN
  \]
A Little English

• An verb can be the word loves, hates or eats:

\[
\begin{align*}
V &\rightarrow \text{loves} \\
V &\rightarrow \text{hates} \\
V &\rightarrow \text{eats}
\end{align*}
\]

A sentence can be a noun phrase, followed by a verb, followed by another noun phrase:

\[
S \rightarrow \text{PVP}
\]
The Little English Grammar

• Taken all together, a grammar $G_1$ for a small subset of unpunctuated English:

<table>
<thead>
<tr>
<th>Production</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow PVP$</td>
<td>$A \rightarrow a$</td>
</tr>
<tr>
<td>$P \rightarrow AN$</td>
<td>$A \rightarrow$ the</td>
</tr>
<tr>
<td>$V \rightarrow$ loves</td>
<td>$N \rightarrow$ dog</td>
</tr>
<tr>
<td>$V \rightarrow$ hates</td>
<td>$N \rightarrow$ cat</td>
</tr>
<tr>
<td>$V \rightarrow$ eats</td>
<td>$N \rightarrow$ rat</td>
</tr>
</tbody>
</table>

• Each *production* says how to modify strings by substitution
• $x \rightarrow y$ says, substring $x$ may be replaced by $y$
Start from $S$ and follow the productions of $G_1$

This can derive a variety of (unpunctuated) English sentences:

$S \Rightarrow PVP \Rightarrow ANVP \Rightarrow theNVP \Rightarrow thecatVP \Rightarrow thecateatsP \Rightarrow thecateatsAN$
$\Rightarrow thecateatsaN \Rightarrow thecateatsarat$

$S \Rightarrow PVP \Rightarrow ANVP \Rightarrow aNVP \Rightarrow adogVP \Rightarrow adoglovesP \Rightarrow adoglovesAN$
$\Rightarrow adoglovestheN \Rightarrow adoglovesthecat$

$S \Rightarrow PVP \Rightarrow ANVP \Rightarrow theNVP \Rightarrow thecatVP \Rightarrow thecathatesP \Rightarrow thecathatesAN$
$\Rightarrow thecathatestheN \Rightarrow thecathatesthedog$
Often there is more than one place in a string where a production could be applied.

For example, $PlovesP$:

- $PlovesP \Rightarrow ANlovesP$
- $PlovesP \Rightarrow PlovesAN$

The derivations on the previous slide chose the leftmost substitution at every step, but that is not a requirement.

The language defined by a grammar is the set of lowercase strings that have at least one derivation from the start symbol $S$.
• Often, a grammar contains more than one production with the same left-hand side
• Those productions can be written in a compressed form
• The grammar is not changed by this
• This example still has ten productions

S → PVP
P → AN
V → loves | hates | eats
A → a | the
N → dog | cat | rat
Informal Definition

A *grammar* is a set of productions of the form $x \rightarrow y$. The strings $x$ and $y$ can contain both lowercase and uppercase letters; $x$ cannot be empty, but $y$ can be $\varepsilon$. One uppercase letter is designated as the start symbol (conventionally, it is the letter $S$).

- Productions define permissible string substitutions
- When a sequence of permissible substitutions starting from $S$ ends in a string that is all lowercase, we say the grammar generates that string
- $L(G)$ is the set of all strings generated by grammar $G$
That final production for $X$ says that $X$ may be replaced by the empty string, so that for example $abbX \Rightarrow abb$

Written in the more compact way, this grammar is:

\[
\begin{align*}
S & \rightarrow aS | X \\
X & \rightarrow bX | \epsilon
\end{align*}
\]
\[ S \rightarrow aS \mid X \]
\[ X \rightarrow bX \mid \varepsilon \]

\[
S \Rightarrow aS \Rightarrow aX \Rightarrow a \\
S \Rightarrow X \Rightarrow bX \Rightarrow b \\
S \Rightarrow aS \Rightarrow aX \Rightarrow abX \Rightarrow abbX \Rightarrow abb \\
S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow aaaX \Rightarrow aaabX \Rightarrow aaabbbX \Rightarrow aaabbb
\]
• For this grammar, all derivations of lowercase strings follow this simple pattern:
  – First use $S \rightarrow aS$ zero or more times
  – Then use $S \rightarrow X$ once
  – Then use $X \rightarrow bX$ zero or more times
  – Then use $X \rightarrow \varepsilon$ once

• So the generated string always consists of zero or more $a$s followed by zero or more $b$s

• $L(G) = L(a^*b^*)$
Untapped Power

- All our examples have used productions with a single uppercase letter on the left-hand side
- Grammars can have any non-empty string on the left-hand side
- The mechanism of substitution is the same
  - $Sb \rightarrow bS$ says that $bS$ can be substituted for $Sb$
- Such productions can be very powerful, but we won't need that power yet
- We'll concentrate on grammars with one uppercase letter on the left-hand side of every production
Outline

• 10.1 A Grammar Example for English
• 10.2 The 4-Tuple
• 10.3 The Language Generated by a Grammar
• 10.4 Every Regular Language Has a Grammar
• 10.5 Right-Linear Grammars
• 10.6 Every Right-linear Grammar Generates a Regular Language
Formalizing Grammars

• Our informal definition relied on the difference between lowercase and uppercase.
• The formal definition will use two separate alphabets:
  – The *terminal symbols* (typically lowercase)
  – The *nonterminal symbols* (typically uppercase)
• So a formal grammar has four parts…
4-Tuple Definition

- A grammar $G$ is a 4-tuple $G = (V, \Sigma, S, P)$, where:
  - $V$ is an alphabet, the *nonterminal alphabet*
  - $\Sigma$ is another alphabet, the *terminal alphabet*, disjoint from $V$
  - $S \in V$ is the *start symbol*
  - $P$ is a finite set of productions, each of the form $x \rightarrow y$, where $x$ and $y$ are strings over $\Sigma \cup V$ and $x \neq \varepsilon$
Example

\[
\begin{align*}
S & \rightarrow aS \mid X \\
X & \rightarrow bX \mid \\
\epsilon & \\
\end{align*}
\]

- Formally, this is \( G = (V, \Sigma, S, P) \), where:
  - \( V = \{S, X\} \)
  - \( \Sigma = \{a, b\} \)
  - \( P = \{S \rightarrow aS, S \rightarrow X, X \rightarrow bX, X \rightarrow \epsilon\} \)
- The order of the 4-tuple is what counts:
  - \( G = (\{S, X\}, \{a, b\}, S, \{S \rightarrow aS, S \rightarrow X, X \rightarrow bX, X \rightarrow \epsilon\}) \)
Outline

• 10.1 A Grammar Example for English
• 10.2 The 4-Tuple
• 10.3 The Language Generated by a Grammar
• 10.4 Every Regular Language Has a Grammar
• 10.5 Right-Linear Grammars
• 10.6 Every Right-linear Grammar Generates a Regular Language
The Program

• For DFAs, we derived a zero-or-more-step $\delta^*$ function from the one-step $\delta$
• For NFAs, we derived a one-step relation on IDs, then extended it to a zero-or-more-step relation
• We'll do the same kind of thing for grammars…
$W \Rightarrow Z$

- Defined for a grammar $G = (V, \Sigma, S, P)$
- $\Rightarrow$ is a relation on strings
- $w \Rightarrow z$ ("$w$ derives $z$") if and only if there exist strings $u, x, y,$ and $v$ over $\Sigma \cup V$, with
  - $w = uxv$
  - $z = uyv$
  - $(x \rightarrow y) \in P$
- That is, $w$ can be transformed into $z$ using one of the substitutions permitted by $G$
Derivations And $w \Rightarrow^* z$

- A sequence of $\Rightarrow$-related strings $x_0 \Rightarrow x_1 \Rightarrow \ldots \Rightarrow x_n$, is an $n$-step derivation.
- $w \Rightarrow^* z$ if and only if there is a derivation of 0 or more steps that starts with $w$ and ends with $z$.
- That is, $w$ can be transformed into $z$ using a sequence of zero or more of the substitutions permitted by $G$. 

*Formal Language*, chapter 10, slide 22
$L(G)$

- The language generated by a grammar $G$ is
  $L(G) = \{x \in \Sigma^* \mid S \Rightarrow^* x\}$
- That is, the set of fully terminal strings derivable from the start symbol
- Notice the restriction $x \in \Sigma^*$:
  - The intermediate strings in a derivation can use both $\Sigma$ and $V$
  - But only the fully terminal strings are in $L(G)$
Outline

• 10.1 A Grammar Example for English
• 10.2 The 4-Tuple
• 10.3 The Language Generated by a Grammar
• 10.4 Every Regular Language Has a Grammar
• 10.5 Right-Linear Grammars
• 10.6 Every Right-linear Grammar Generates a Regular Language
NFA to Grammar

• To show that there is a grammar for every regular language, we will show how to convert any NFA into an equivalent grammar
• That is, given an NFA $M$, construct a grammar $G$ with $L(M) = L(G)$
• First, an example…
Example:

- The grammar we will construct generates $L(M)$
- In fact, its derivations will mimic what $M$ does
- For each state, our grammar will have a nonterminal symbol ($S$, $R$ and $T$)
- The start state will be the grammar's start symbol
- The grammar will have one production for each transition of the NFA, and one for each accepting state
Example:

- For each possible transition $Y \in \delta(X,z)$ in the NFA, our grammar has a production $X \rightarrow zY$
- That gives us these four to start with:

<table>
<thead>
<tr>
<th>Transition of $M$</th>
<th>Production in $G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta(S,a) = {S}$</td>
<td>$S \rightarrow aS$</td>
</tr>
<tr>
<td>$\delta(S,b) = {R}$</td>
<td>$S \rightarrow bR$</td>
</tr>
<tr>
<td>$\delta(R,c) = {R}$</td>
<td>$R \rightarrow cR$</td>
</tr>
<tr>
<td>$\delta(R,\epsilon) = {T}$</td>
<td>$R \rightarrow T$</td>
</tr>
</tbody>
</table>
Example:

- In addition, for each accepting state in the NFA, our grammar has an $\varepsilon$-production.
- That adds one more:

<table>
<thead>
<tr>
<th>Accepting state of $M$</th>
<th>Production in $G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>$T \rightarrow \varepsilon$</td>
</tr>
</tbody>
</table>
Example:

- The complete grammar has one production for each transition, and one for each accepting state:

\[
\begin{align*}
S &\rightarrow aS \\
S &\rightarrow bR \\
R &\rightarrow cR \\
R &\rightarrow T \\
T &\rightarrow \varepsilon 
\end{align*}
\]
Compare the behavior of $M$ as it accepts $abc$ with the behavior of $G$ as it generates $abc$:

\[(S, abc) \Rightarrow (S, b) \Rightarrow (R, c) \Rightarrow (R, \epsilon) \Rightarrow (T, \epsilon)\]

\[
\begin{align*}
S & \Rightarrow aS \\
S & \Rightarrow bR \\
R & \Rightarrow cR \\
R & \Rightarrow T \\
T & \Rightarrow \epsilon
\end{align*}
\]

- Every time the NFA reads a symbol, the grammar generates that symbol.
- In general, $M$ can be in state $Y$ after reading string $x$ if and only if $G$ can derive the string $xY$.
Theorem 10.4
Every regular language is generated by some grammar.

- Proof is by construction; let \( M = (Q, \Sigma, \delta, S, F) \) be any NFA
- Construct \( G = (Q, \Sigma, S, P) \)
  - \( Q, \Sigma, \) and \( S \) are the same as for \( M \)
  - \( P \) is constructed from \( \delta \) and \( F \):
    - Wherever \( M \) has \( Y \in \delta(X,z) \), \( P \) contains \( X \rightarrow zY \)
    - And for each \( X \in F \), \( P \) contains \( X \rightarrow \varepsilon \)
- Now \( G \) has \( X \rightarrow zY \) whenever \( (X, z) \mapsto (Y, \varepsilon) \)
- By induction we can extend this to any string \( z \in \Sigma^* \):
  - \( G \) has \( X \rightarrow^* zY \) whenever \( (X, z) \mapsto^* (Y, \varepsilon) \)
  - And by construction, \( G \) has \( Y \rightarrow \varepsilon \) whenever \( M \) has \( Y \in F \)
- So for all strings \( z \in \Sigma^* \), \( \delta^*(S,z) \) contains at least one element of \( F \) if and only if \( S \Rightarrow^* z \)
- \( L(M) = L(G) \)
Outline

• 10.1 A Grammar Example for English
• 10.2 The 4-Tuple
• 10.3 The Language Generated by a Grammar
• 10.4 Every Regular Language Has a Grammar
• 10.5 Right-Linear Grammars
• 10.6 Every Right-linear Grammar Generates a Regular Language
Single-Step Grammars

- A grammar $G = (V, \Sigma, S, P)$ is *single step* if and only if every production in $P$ is in one of these three forms, where $X \in V$, $Y \in V$, and $z \in \Sigma$:
  - $X \rightarrow zY$
  - $X \rightarrow z$
  - $X \rightarrow \varepsilon$

- Given any single-step grammar, we could run the previous construction backwards, building an equivalent NFA…
Reverse Example

- This grammar generates \( L(ab^*a) \):
- All its productions are of the kinds built in our construction
- Running the construction backwards, we get three states \( S, R, \) and \( T \)
- The first three productions give us the three arrows, and the fourth makes \( T \) accepting:

\[
\begin{align*}
S & \rightarrow aR \\
R & \rightarrow bR \\
R & \rightarrow aT \\
T & \rightarrow \varepsilon
\end{align*}
\]
Production Massage

S → abR
R → a

• Even if all the productions are not of the required form, it is sometimes possible to massage them until they are

• \( S \rightarrow abR \) does not have the right form:
  – Equivalent productions \( S \rightarrow aX \) and \( X \rightarrow bR \) do

• \( R \rightarrow a \) does not have the right form:
  – Equivalent productions \( R \rightarrow aY \) and \( Y \rightarrow \varepsilon \) do

• After those changes we can run the construction backwards…
Massaged Reverse Example

\[ S \rightarrow abR \]
\[ R \rightarrow a \]

\[ S \rightarrow aX \]
\[ X \rightarrow bR \]
\[ R \rightarrow aY \]
\[ Y \rightarrow \varepsilon \]
Right-Linear Grammars

- A grammar $G = (V, \Sigma, S, P)$ is right linear if and only if every production in $P$ is in one of these two forms, where $X \in V$, $Y \in V$, and $z \in \Sigma^*$:
  - $X \rightarrow zY$, or
  - $X \rightarrow z$
- So every production has:
  - A single nonterminal on the left
  - At most one nonterminal on the right, and only as the rightmost symbol
- Note that this includes all single-step grammars
- This special form makes it easy to massage the productions and then transform them into NFAs
Lemma 10.5

Every right-linear grammar $G$ is equivalent to some single-step grammar $G'$.

- Proof is by construction
- Let $G = (V, \Sigma, S, P)$ be any right-linear grammar
- Each production is $X \rightarrow z_1...z_n\omega$, where $\omega \in V$ or $\omega = \varepsilon$
- For each such production, let $P$ contains these $n+1$ productions, where each $K_i$ is a new nonterminal symbol:
  - Now let $G = (V', \Sigma, S, P')$, where $V'$ is the set of nonterminals used in $P'$
  - Any step of a derivation $G$ is equivalent to the corresponding $n+1$ steps in $G'$
  - The reverse is true for derivations of terminal strings in $G'$
  - So $L(G) = L(G')$
Outline

• 10.1 A Grammar Example for English
• 10.2 The 4-Tuple
• 10.3 The Language Generated by a Grammar
• 10.4 Every Regular Language Has a Grammar
• 10.5 Right-Linear Grammars
• 10.6 Every Right-linear Grammar Generates a Regular Language
Theorem 10.6

For every right-linear grammar $G$, $L(G)$ is regular.

- Proof is by construction
- Use Lemma 10.5 to get single-step form, then use the reverse of the construction from Theorem 10.4
Left-Linear Grammars

• A grammar $G = (V, \Sigma, S, P)$ is left linear if and only if every production in $P$ is in one of these two forms, where $X \in V$, $Y \in V$, and $z \in \Sigma^*$:
  - $X \rightarrow Yz$, or
  - $X \rightarrow z$

• This parallels the definition of right-linear

• With a little more work, one can show that the language generated is also always regular
Regular Grammars, Regular Languages

- Grammars that are either left-linear or right-linear are called regular grammars.
- A simple inspection tells you whether $G$ is a regular grammar; if it is, $L(G)$ is a regular language.
- Note that if $G$ is not a regular grammar, that tells you nothing: $L(G)$ might still be regular language.
- This example is not right-linear and not left-linear, but $L(G)$ is the regular language $L((aaa)^*)$:

\[
S \rightarrow aSaa \mid \varepsilon
\]
The Next Big Question

- We know that all regular grammars generate regular languages
- We've seen a non-regular grammar that still generates a regular language
- So are there any grammars that generate languages that are not regular?
- For that matter, do any non-regular languages exist?
- Answers to these in the next chapter