Chapter Twelve:
Context-Free Languages

Context-Free Languages

- We defined the right-linear grammars by giving a simple restriction on the form of each production.
- By relaxing that restriction a bit, we get a broader class of grammars: the *context-free grammars*.
- These grammars generate the context-free languages, which include all the regular languages along with many that are not regular.

Outline

- 12.1 Context-Free Grammars and Languages
- 12.2 Writing CFGs
- 12.3 CFG Applications: BNF
- 12.4 Parse Trees
- 12.5 Ambiguity
- 12.6 EBNF

Examples

- We can prove that these languages are not regular, yet they have grammars
  - $\{a^n b^n\}$ $\quad S \rightarrow aSb \mid \varepsilon$
  - $\{xx^R \mid x \in \{a,b\}^*\}$ $\quad S \rightarrow aSa \mid bSb \mid \varepsilon$
  - $\{a^n b^j a^n \mid n \geq 0, j \geq 1\}$ $\quad S \rightarrow aSa \mid R$, $R \rightarrow bR \mid b$
- Although not right-linear, these grammars still follow a rather restricted form…

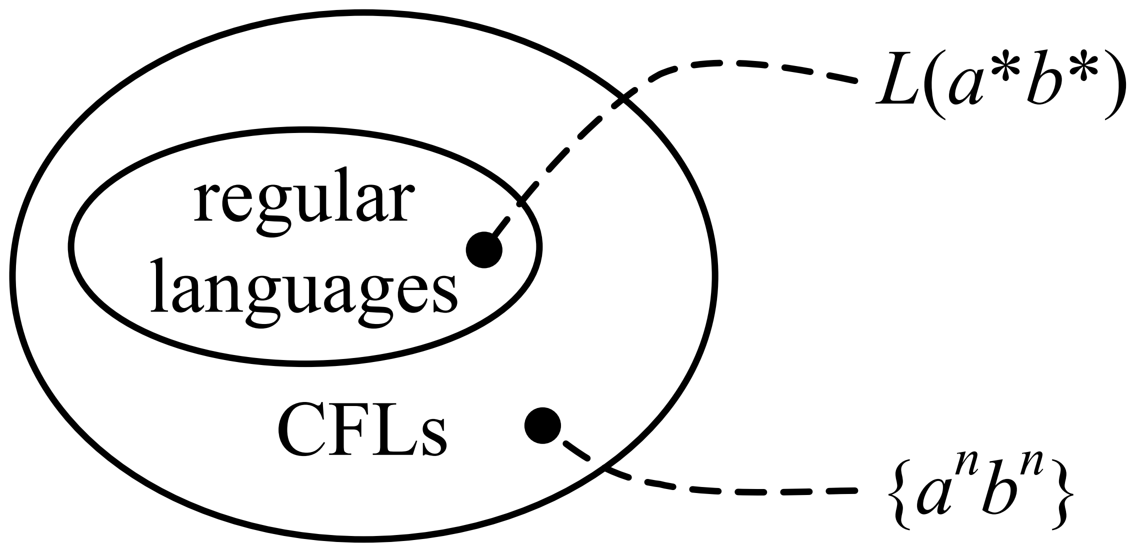
Context-Free Grammars

- A context-free grammar (CFG) is one in which every production has a single nonterminal symbol on the left-hand side
- A production like $R \rightarrow y$ is permitted
  - It says that *R* can be replaced with *y*, regardless of the context of symbols around *R* in the string
- One like $uRz \rightarrow uyz$ is not permitted
  - That would be context-sensitive: it says that *R* can be replaced with *y* only in a specific context

Context-Free Languages

- A context-free language (CFL) is one that is $L(G)$ for some CFG G
- Every regular language is a CFL
  - Every regular language has a right-linear grammar
  - Every right-linear grammar is a CFG
- But not every CFL is regular
  - $\{a^n b^n\}$
  - $\{xx^R \mid x \in \{a,b\}^*\}$
  - $\{a^n b^j a^n \mid n \geq 0, j \geq 1\}$

Language Classes So Far



Outline

- 12.1 Context-Free Grammars and Languages
- 12.2 Writing CFGs
- 12.3 CFG Applications: BNF
- 12.4 Parse Trees
- 12.5 Ambiguity
- 12.6 EBNF

Writing CFGs

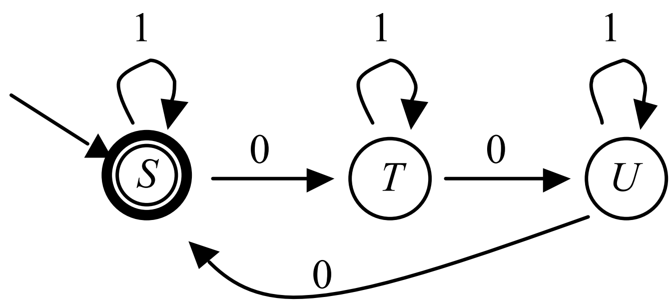
- Programming:
  - A program is a finite, structured, mechanical thing that specifies a potentially infinite collection of runtime behaviors
  - You have to imagine how the code you are crafting will unfold when it executes
- Writing grammars:
  - A grammar is a finite, structured, mechanical thing that specifies a potentially infinite language
  - You have to imagine how the productions you are crafting will unfold in the derivations of terminal strings
- Programming and grammar-writing use some of the same mental muscles
- Here follow some techniques and examples…

Regular Languages

- If the language is regular, we already have a technique for constructing a CFG
  - Start with an NFA
  - Convert to a right-linear grammar using the construction from chapter 10

Example

$L = \{x \in \{0,1\}^* \mid$ the number of 0s in x is divisible by 3$\}$



$$S \rightarrow 1S \mid 0T \mid \varepsilon$$
$$T \rightarrow 1T \mid 0U$$
$$U \rightarrow 1U \mid 0S$$

Example

$L = \{x \in \{0,1\}^* \mid \text{the number of 0s in } x \text{ is divisible by 3}\}$

- The conversion from NFA to grammar always works
- But it does not always produce a pretty grammar
- It may be possible to design a smaller or otherwise more readable CFG manually:

$$S \rightarrow 1S \mid 0T \mid \varepsilon$$
$$T \rightarrow 1T \mid 0U$$
$$U \rightarrow 1U \mid 0S$$

$$S \rightarrow T0T0T0S \mid T$$
$$T \rightarrow 1T \mid \varepsilon$$

Balanced Pairs

- CFLs often seem to involve balanced pairs
  - $\{a^n b^n\}$: every *a* paired with *b* on the other side
  - $\{xx^R \mid x \in \{a,b\}^*\}$: each symbol in *x* paired with its mirror image in x^R
  - $\{a^n b^j a^n \mid n \geq 0, j \geq 1\}$: each *a* on the left paired with one on the right
- To get matching pairs, use a recursive production of the form $R \rightarrow xRy$
- This generates any number of *x*s, each of which is matched with a *y* on the other side

Examples

- We've seen these before:
  - $\{a^n b^n\}$ $S \rightarrow aSb \mid \varepsilon$
  - $\{xx^R \mid x \in \{a,b\}^*\}$ $S \rightarrow aSa \mid bSb \mid \varepsilon$
  - $\{a^n b^j a^n \mid n \geq 0, j \geq 1\}$ $S \rightarrow aSa \mid R$
$R \rightarrow bR \mid b$
- Notice that they all use the $R \rightarrow xRy$ trick

Examples

- $\{a^n b^{3n}\}$
  - Each *a* on the left can be paired with three *b*s on the right
  - That gives

$$S \rightarrow aSbbb \mid \varepsilon$$

- $\{xy \mid x \in \{a,b\}^*,\ y \in \{c,d\}^*, \text{ and } |x| = |y|\}$
  - Each symbol on the left (either *a* or *b*) can be paired with one on the right (either *c* or *d*)
  - That gives

$$\begin{aligned} S &\rightarrow XSY \mid \varepsilon \\ X &\rightarrow a \mid b \\ Y &\rightarrow c \mid d \end{aligned}$$

Concatenations

- A divide-and-conquer approach is often helpful
- For example, $L = \{a^n b^n c^m d^m\}$
  - We can make grammars for $\{a^n b^n\}$ and $\{c^m d^m\}$:

$$S_1 \rightarrow aS_1b \mid \varepsilon$$

$$S_2 \rightarrow cS_2d \mid \varepsilon$$

  - Now every string in L consists of a string from the first followed by a string from the second
  - So combine the two grammars and add a new start symbol:

$$S \rightarrow S_1S_2$$
$$S_1 \rightarrow aS_1b \mid \varepsilon$$
$$S_2 \rightarrow cS_2d \mid \varepsilon$$

Concatenations, In General

- Sometimes a CFL L can be thought of as the concatenation of two languages L_1 and L_2
  - That is, $L = L_1L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$
- Then you can write a CFG for L by combining separate CFGs for L_1 and L_2
  - Be careful to keep the two sets of nonterminals separate, so no nonterminal is used in both
  - In particular, use two separate start symbols S_1 and S_2
- The grammar for L consists of all the productions from the two sub-grammars, plus a new start symbol S with the production $S \rightarrow S_1S_2$

Unions, In General

- Sometimes a CFL L can be thought of as the union of two languages $L = L_1 \cup L_2$
- Then you can write a CFG for L by combining separate CFGs for L_1 and L_2
  - Be careful to keep the two sets of nonterminals separate, so no nonterminal is used in both
  - In particular, use two separate start symbols S_1 and S_2
- The grammar for L consists of all the productions from the two sub-grammars, plus a new start symbol S with the production $S \rightarrow S_1 \mid S_2$

Example

$L = \{z \in \{a,b\}^* \mid z = xx^R$ for some x, or $|z|$ is odd$\}$

- This can be thought of as a union: $L = L_1 \cup L_2$
  - $L_1 = \{xx^R \mid x \in \{a,b\}^*\}$

$$S_1 \rightarrow aS_1a \mid bS_1b \mid \varepsilon$$

  - $L_2 = \{z \in \{a,b\}^* \mid \ |z| \text{ is odd}\}$

$$S_2 \rightarrow XXS_2 \mid X$$
$$X \rightarrow a \mid b$$

- So a grammar for L is

$$S \rightarrow S_1 \mid S_2$$
$$S_1 \rightarrow aS_1a \mid bS_1b \mid \varepsilon$$
$$S_2 \rightarrow XXS_2 \mid X$$
$$X \rightarrow a \mid b$$

Example

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

- This can be thought of as a union:
  - $L = \{a^n b^m \mid n < m\} \cup \{a^n b^m \mid n > m\}$
- Each of those two parts can be thought of as a concatenation:
  - $L_1 = \{a^n b^n\}$
  - $L_2 = \{b^i \mid i > 0\}$
  - $L_3 = \{a^i \mid i > 0\}$
  - $L = L_1 L_2 \cup L_3 L_1$
- The resulting grammar:

$$\begin{aligned}
S &\rightarrow S_1 S_2 \mid S_3 S_1 \\
S_1 &\rightarrow a S_1 b \mid \varepsilon \\
S_2 &\rightarrow b S_2 \mid b \\
S_3 &\rightarrow a S_3 \mid a
\end{aligned}$$