CHAPTER 2

Context-Free Languages

Contents

Context-Free Grammars: an overview

- ***Context-free grammars*** is a more powerful method of describing languages.
- Such grammars can describe certain features that have a *recursive structure* which makes them useful in a variety of applications.
- The collection of languages associated with context-free grammars are called ***the context-free languages***.
- They include all the regular languages and many additional languages.
- We will give a formal definition of context-free grammars and study the properties of context-free languages.
- We will also introduce ***pushdown automata***, a class of machines recognizing the context-free languages.

Context-Free Grammars

- Consider the following example of a context-free grammar, call it *G1*.

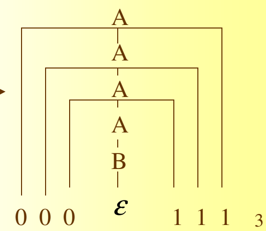
$$A \to 0A1$$
$$A \to B$$
$$B \to \varepsilon$$

- A grammar consists of a collection of ***substitution rules***, also called ***productions***.
- Each rule appears as a line in the grammar and comprises a ***symbol*** and a ***string***, separated by an ***arrow***.

- The symbol is called a ***variable*** (capital letters; A,B). The string consists of variables and other symbols called ***terminals*** (lowercase letters, numbers, or special symbols; 0,1,ε).
- One variable is designated the ***start variable***. (usually, the variable on the left-hand side of the topmost rule; A).
- We use grammars to describe a language by generating each string of that language.
  - For example, grammar *G1* generates the string 000111
  - The sequence of substitutions to obtain a string is called a ***derivation.***
  - A derivation of string 000111 in grammar *G1* is

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000111$$

(this can be shown also by a ***parse tree***)



- All strings generated in this way constitute the ***language of the grammar** G1, L(G1)*.
- It is clear that *L(G1)* is $\{0^m 1^n : \; n = m\}$.

Context-Free Grammars (cont.)

- Any language that can be generated by some context-free grammar is called a ***context-free language*** (CFL)
- For convenience when presenting a context-free grammar, we abbreviate several rules with the same left-hand variable, such as *A→0A1* and *A→B*, into a single line *A→ 0A1* | *B*, using the symbol “|” as an “or”.
- **Example** of a context-free grammar called *G2*, which describes a fragment of the English language:

$$\langle SENTENCE \rangle \to \langle NOUN-PHRASE \rangle \langle VERB-PHRASE \rangle$$
$$\langle NOUN-PHRASE \rangle \to \langle CMPLX-NOUN \rangle | \langle CMPLX-NOUN \rangle \langle PREP-PHRASE \rangle$$
$$\langle VERB-PHRASE \rangle \to \langle CMPLX-VERB \rangle | \langle CMPLX-VERB \rangle \langle PREP-PHRASE \rangle$$
$$\langle PREP-PHRASE \rangle \to \langle PREP \rangle \langle CMPLX-NOUN \rangle$$
$$\langle CMPLX-NOUN \rangle \to \langle ARTICLE \rangle \langle NOUN \rangle$$
$$\langle CMPLX-VERB \rangle \to \langle VERB \rangle | \langle VERB \rangle \langle NOUN-PHRASE \rangle$$
$$\langle ARTICLE \rangle \to a | the$$
$$\langle NOUN \rangle \to boy | girl | flower$$
$$\langle VERB \rangle \to touches | likes | sees$$
$$\langle PREP \rangle \to with$$

- Strings in *L(G2)* include the following three examples

a boy sees
the boy sees a flower
a girl with a flower likes the boy

- Each of these strings has a derivation in grammar *G2*. The following is a derivation of the first string on the list

$$\langle SENTENCE \rangle \Rightarrow \langle NOUN-PHRASE \rangle \langle VERB-PHRASE \rangle \Rightarrow \langle CMPLX-NOUN \rangle \langle VERB-PHRASE \rangle$$
$$\Rightarrow \langle ARTICLE \rangle \langle NOUN \rangle \langle VERB-PHRASE \rangle \Rightarrow a \langle NOUN \rangle \langle VERB-PHRASE \rangle \Rightarrow$$
$$a\ boy \langle VERB-PHRASE \rangle \Rightarrow a\ boy \langle CMPLX-VERB \rangle \Rightarrow a\ boy \langle VERB \rangle \Rightarrow a\ boy\ sees$$

Formal Definition of a Context-Free Grammar

- A context-free grammar is a 4-tuple (V, Σ, R, S), where

V	is a finite set called ***variables***,
Σ	is a finite set (=alphabet), disjoint from *V,* called the ***terminals,***
R	is a finite set of ***rules,*** with each rule being a variable and a string of variables and terminals, and
$S \in V$	is the ***start variable***.

- If *u, v,* and *w* are strings of variables and terminals, and $A \rightarrow w$ is a rule of the grammar, we say that *uAv* yields uwv, writing $uAv \Rightarrow uwv$.
- Write $u \overset{*}{\Rightarrow} v$ if *u=v* or if a sequence $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ and

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v.$$

- The ***language of the grammar*** is $\{w \in \Sigma^* : S \overset{*}{\Rightarrow} w\}$.
- Hence, for *G1*: *V={A,B},* $\Sigma = \{0, 1, \varepsilon\}$, *S=A,* and *R* is the collection of those three rules.

  for *G2*: V={<SENTENCE>,<NOUN-PHRASE>,<VERB-PHRASE>,<PREP-PHRASE>, <CMPLX-NOUN>,<CMPLX-VERB>,<ARTICLE>,<NOUN>,<VERB>, <PREP>}, $\Sigma = \{a, b, c, \ldots, z, " \ "\}$.
- Example: *G3=({S},{(,)},R,S).* The set of rules is $S \rightarrow (S) \mid SS \mid \varepsilon$.

  L(G3) is the language of all strings of properly nested parentheses.

Designing Context-Free Grammars

- The design of context-free grammars requires creativity (no simple universal methods).
- The following techniques will be helpful, singly or in combination, when you are faced with the problem of constructing a CFG.

  a) Many CFGs are the union of simpler CFGs. If you must construct a CFG for a CFL that you can break into simpler pieces, do so and then construct individual grammars for each piece. These individual grammars can be easily combined into a grammar for the original language by putting all their rules together and then adding the new rule $S \rightarrow S_1 \mid S_2 \mid \ldots \mid S_k$, where the variables S_i are the start variables for the individual grammars. Solving several simpler problems is often easier than solving one complicated problem.

  To get a grammar for $\{0^n 1^n : n \geq 0\} \cup \{1^n 0^n : n \geq 0\}$

  first construct two grammars

  $$S_1 \rightarrow 0S_1 1 \mid \varepsilon \quad \textit{for} \ \{0^n 1^n : n \geq 0\} \quad \textit{and}$$
  $$S_2 \rightarrow 1S_2 0 \mid \varepsilon \quad \textit{for} \ \{1^n 0^n : n \geq 0\},$$

  and then add the rule $S \rightarrow S_1 \mid S_2$ to give the grammar

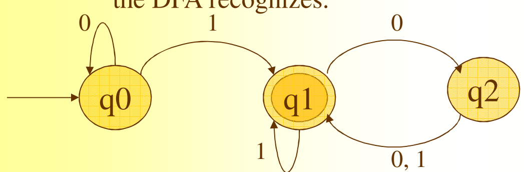
  $$S \rightarrow S_1 \mid S_2; \qquad S_1 \rightarrow 0S_1 1 \mid \varepsilon; \qquad S_2 \rightarrow 1S_2 0 \mid \varepsilon$$

Designing Context-Free Grammars (cont.)

b) Constructing a CFG for a language that happens to be regular is easy if you can construct a DFA for that language. You can convert any DFA into an equivalent CFG as follows:

- Make a variable R_i for each state q_i of the DFA.
- Add rule $R_i \to aR_j$ to the CRG if there is an arc from q_i to q_j with label *a*.
- Add the rule $R_i \to \varepsilon$ if q_i is an accept state of the DFA.
- Make R_0 the start variable of the grammar, where q_0 is the start state of the machine.

Verify on your own that the resulting CFG generates the same language that the DFA recognizes.



$$R_0 \to 0R_0 \mid 1R_1$$
$$R_1 \to 1R_1 \mid 0R_2 \mid \varepsilon$$
$$R_2 \to 0R_1 \mid 1R_1$$

$=\{$w: w contains at least one 1 and an even number of 0s follow the last 1$\}$

- Thus, **any regular language is a CFL.**

Designing Context-Free Grammars (cont.)

c) Use the rule of the form $R \to uRv$ if context-free languages contain strings with two substrings that are 'linked' in the sense that a machine for such a language would need to remember an unbounded amount of information about one of the substrings to verify that it corresponds properly to the other substring.

this situation occurs in the language $\{0^m 1^n : \ n = m\}$.

c) In more complex languages, the strings may contain certain structures that appear recursively as part of other (or the same) structures.

this situation occurs in the language of all strings of properly nested parentheses.

$$S \to (S) \mid SS \mid \varepsilon.$$

the situation occurs also in grammar that generates arithmetic expressions.

$$G_4: \quad \begin{aligned} <EXPR> &\to <EXPR> + <TERM> \mid <TERM> \\ <TERM> &\to <TERM> \times <FACTOR> \mid <FACTOR> \\ <FACTOR> &\to (<EXPR>) \mid a \end{aligned}$$

$$G_5: \quad <EXPR> \to <EXPR> + <EXPR> \mid <EXPR> \times <EXPR> \mid (<EXPR>) \mid a$$

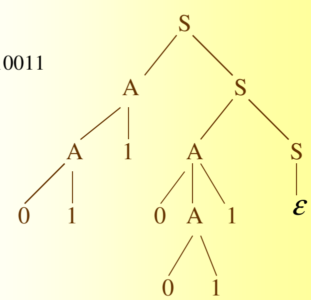
- Place the variable symbol generating the structure in the location of the rules corresponding to where that structure may recursively appear.

Leftmost and Rightmost Derivations

- We have a choice of variable to replace at each step.
  - derivations may appear different only because we make the same replacement in a different order.
  - to avoid such differences, we may restrict the choice.
- A *leftmost derivation* always replace the leftmost variable in a string.
- A *rightmost derivation* always replace the rightmost variable in a string.
- $\underset{lm}{\Rightarrow}, \underset{rm}{\Rightarrow}$ used to indicate derivations are leftmost or rightmost.
- Example: strings of 0's and 1's such that each block of 0's is followed by at least as many 1's.

$S \to AS \mid \varepsilon$
$A \to 0A1 \mid A1 \mid 01$

- $S \underset{lm}{\Rightarrow} AS \underset{lm}{\Rightarrow} A1S \underset{lm}{\Rightarrow} 011S \underset{lm}{\Rightarrow} 011AS \underset{lm}{\Rightarrow} 0110A1S \underset{lm}{\Rightarrow} 0110011S \underset{lm}{\Rightarrow} 0110011$
- $S \underset{rm}{\Rightarrow} AS \underset{rm}{\Rightarrow} AAS \underset{rm}{\Rightarrow} AA \underset{rm}{\Rightarrow} A0A1 \underset{rm}{\Rightarrow} A0011 \underset{rm}{\Rightarrow} A10011 \underset{rm}{\Rightarrow} 0110011$



One can prove the following for a grammar G

$S \overset{*}{\Rightarrow} w$ (*i.e.*, w *is in* $L(G)$) *iff*
$S \overset{*}{\underset{lm}{\Rightarrow}} w$ *iff*
$S \overset{*}{\underset{rm}{\Rightarrow}} w$ *iff*
there is a parse tree for G with root S and yield (labels of leaves, from the left) w.

Ambiguous Grammars

- A CFG *G* is ***ambiguous*** if one or more words from *L(G)* have multiple leftmost derivations from the start variable.
  - equivalently: multiple rightmost derivations, or multiple parse trees.
- Example: consider $S \to AS \mid \varepsilon; \quad A \to 0A1 \mid A1 \mid 01$ and the string 00111.
  { strings of 0's and 1's such that each block of 0's is followed by at least as many 1's }

- $S \underset{lm}{\Rightarrow} AS \underset{lm}{\Rightarrow} 0A1S \underset{lm}{\Rightarrow} 0A11S \underset{lm}{\Rightarrow} 00111S \underset{lm}{\Rightarrow} 00111$
- $S \underset{lm}{\Rightarrow} AS \underset{lm}{\Rightarrow} A1S \underset{lm}{\Rightarrow} 0A11S \underset{lm}{\Rightarrow} 00111S \underset{lm}{\Rightarrow} 00111$

Inherently Ambiguous Languages

- A CFL *L* is ***inherently ambiguous*** if every CFG for *L* is ambiguous.
  - such CFLs exist: e.g., $\{2^i 1^j 2^k : i = j \text{ or } j = k\}$.
  - an inherently ambiguous languages would absolutely unsuitable as a programming language.
- The language of our example grammar is not inherently ambiguous, even though the grammar is ambiguous.
  - Change the grammar to force the extra 1's to be generated last.

$$S \to AS \mid \varepsilon; \quad A \to 0A1 \mid B; \quad B \to B1 \mid 01$$

Chomsky Normal Form

- A context-free grammar is in ***Chomsky form*** if every rule is of the form $\begin{array}{l} A \to BC \\ A \to a \end{array}$

where a is any terminal and A,B,C are any variables- except that B and C may not be the start variable. In addition we permit the rule $S \to \varepsilon$, where S is the start variable.

Theorem: *Any CFL is generated by a CFG in Chomsky form.*

Proof (by construction; we convert any grammar into Chomsky form)

- add start symbol S_0 and the rule $S_0 \to S$, where S was the original start symbol.
- remove an ε -rule $A \to \varepsilon$, where A is not the start variable (v,u,w are strings of variables and terminals).
  - then for each occurrence of an A on the right-hand side of a rule, add a new rule with that occurrence deleted (e.g., $R \to uAv \succ\!\!\to R \to uv$
    $R \to uAvAw \succ\!\!\to R \to uvAw;\ R \to uAvw;\ R \to uvw.$
  - if we have $R \to A$ we add $R \to \varepsilon$ unless we had previously removed the rule $R \to \varepsilon.$
  - repeat this step until we eliminate all ε - rules not involving the start symbol.
- remove a unite rule $A \to B$. Whenever a rule $B \to u$ appears, add the rule $A \to u$ unless this was a unit rule previously deleted. Repeat.
- replace each rule $A \to u_1 u_2 ... u_k$, where $k \geq 3$ and each u_i is a variable or terminal with rules $A \to u_1 A_1;\ A_1 \to u_2 A_2;\ A_2 \to u_3 A_3;\ ... ;\ A_{k-2} \to u_{k-1} u_k.$ A_i are new variables.

if $k \geq 2,$ replace any terminal u_i in the preceding rule(s) with new variable U_i and add the rule $U_i \to u_i.$

Example

$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \varepsilon$

$\xrightarrow{S_0}$

$S_0 \to S$
$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \varepsilon$

$\xrightarrow{B \to \varepsilon}$

$S_0 \to S$
$S \to ASA \mid aB \mid a$
$A \to B \mid S \mid \varepsilon$
$B \to b$

$\xrightarrow{A \to \varepsilon}$

$S_0 \to S$
$S \to ASA \mid aB \mid a \mid SA \mid AS \mid S$
$A \to B \mid S$
$B \to b$

$\xrightarrow{S \to S}$

$S_0 \to S$
$S \to ASA \mid aB \mid a \mid SA \mid AS$
$A \to B \mid S$
$B \to b$

$\xrightarrow{S_0 \to S}$

$S_0 \to ASA \mid aB \mid a \mid SA \mid AS$
$S \to ASA \mid aB \mid a \mid SA \mid AS$
$A \to B \mid S$
$B \to b$

$\xrightarrow[A \to S]{A \to B}$

$S_0 \to ASA \mid aB \mid a \mid SA \mid AS$
$S \to ASA \mid aB \mid a \mid SA \mid AS$
$A \to b \mid ASA \mid aB \mid a \mid SA \mid AS$
$B \to b$

Final step: we simplified the resulting grammar by using a single variable U and rule $U \to a$

$S_0 \to AA_1 \mid UB \mid a \mid SA \mid AS$
$S \to AA_1 \mid UB \mid a \mid SA \mid AS$
$A \to b \mid AA_1 \mid UB \mid a \mid SA \mid AS$
$A_1 \to SA$
$U \to a$
$B \to b$