# Lecture 19

## Context-Free Grammars and Languages

You may have seen something like the following used to give a formal definition of a language. This notation is sometimes called BNF for *Backus–Naur form.*

<stmt> ::= <if-stmt> | <while-stmt> | <begin-stmt> | <assg-stmt>
<if-stmt> ::= **if** <bool-expr> **then** <stmt> **else** <stmt>
<while-stmt> ::= **while** <bool-expr> **do** <stmt>
<begin-stmt> ::= **begin** <stmt-list> **end**
<stmt-list> ::= <stmt> | <stmt> ; <stmt-list>
<assg-stmt> ::= <var> := <arith-expr>
<bool-expr> ::= <arith-expr><compare-op><arith-expr>
<compare-op> ::= $<$ | $>$ | $\leq$ | $\geq$ | $=$ | $\neq$
<arith-expr> ::= <var> | <const> | (<arith-expr><arith-op><arith-expr>)
<arith-op> ::= $+$ | $-$ | $*$ | /
<const> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<var> ::= $a$ | $b$ | $c$ | $\cdots$ | $x$ | $y$ | $z$

This is an example of a *context-free grammar.* It consists of a finite set of rules defining the set of well-formed expressions in some language; in this example, the syntactically correct programs of a simple PASCAL-like programming language.

For example, the first rule above says that a *statement* is either an *if statement*, a *while statement*, a *begin statement*, or an *assignment statement*. If statements, while statements, begin statements, and assignment statements are described formally by other rules further down. The third rule says that a *while statement* consists of the word **while**, followed by a Boolean expression, followed by the word **do**, followed by a statement.

The objects <xxx> are called *nonterminal symbols*. Each nonterminal symbol generates a set of strings over a finite alphabet $\Sigma$ in a systematic way described formally below. For example, the nonterminal <arith-expr> above generates the set of syntactically correct arithmetic expressions in this language. The strings corresponding to the nonterminal <xxx> are generated using rules with <xxx> on the left-hand side. The alternatives on the right-hand side, separated by vertical bars |, describe different ways strings corresponding to <xxx> can be generated. These alternatives may involve other nonterminals <yyy>, which must be further eliminated by applying rules with <yyy> on the left-hand side. The rules can be recursive; for example, the rule above for <stmt-list> says that a statement list is either a statement or a statement followed by a semicolon (;) followed by a statement list.

The string

$$\textbf{while } x \leq y \textbf{ do begin } x := (x+1)\,;\, y := (y-1) \textbf{ end} \tag{19.1}$$

is generated by the nonterminal <stmt> in the grammar above. To show this, we can give a sequence of expressions called *sentential forms* starting from <stmt> and ending with the string (19.1) such that each sentential form is derived from the previous by an application of one of the rules. Each application consists of replacing some nonterminal symbol <xxx> in the sentential form with one of the alternatives on the right-hand side of a rule for <xxx>. Here are the first few sentential forms in a derivation of (19.1):

<stmt>

<while-stmt>

**while** <bool-expr> **do** <stmt>

**while** <arith-expr><compare-op><arith-expr> **do** <stmt>

**while** <var><compare-op><arith-expr> **do** <stmt>

**while** <var> $\leq$ <arith-expr> **do** <stmt>

**while** <var> $\leq$ <var> **do** <stmt>

**while** $x \leq$ <var> **do** <stmt>

**while** $x \leq y$ **do** <stmt>

**while** $x \leq y$ **do** <begin-stmt>

...

Applying different rules will yield different results. For example, the string

$$\textbf{begin if } z = (x + 3) \textbf{ then } y := z \textbf{ else } y := x \textbf{ end}$$

can also be generated. The set of all strings not containing any nonterminals generated by the grammar is called the *language* generated by the grammar. In general, this set of strings may be infinite, even if the set of rules is finite.

There may also be several different derivations of the same string. A grammar is said to be *unambiguous* if this cannot happen. The grammar given above is unambiguous.

We will give a general definition of context-free grammars (CFGs) and the languages they generate. The language (subset of $\Sigma^*$) generated by the context-free grammar $G$ is denoted $L(G)$. A subset of $\Sigma^*$ is called a *context-free language* (CFL) if it is $L(G)$ for some CFG $G$.

CFLs are good for describing infinite sets of strings in a finite way. They are particularly useful in computer science for describing the syntax of programming languages, well-formed arithmetic expressions, well-nested begin-end blocks, strings of balanced parentheses, and so on.

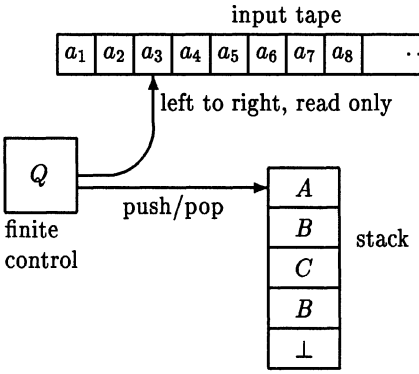
All regular sets are CFLs (Homework 5, Exercise 1), but not necessarily vice versa. The following are examples of CFLs that are not regular:

- $\{a^n b^n \mid n \geq 0\}$;
- $\{\text{palindromes over } \{a, b\}\} = \{x \in \{a, b\}^* \mid x = \textbf{rev } x\}$; and
- {balanced strings of parentheses}.

Not all sets are CFLs; for example, the set $\{a^n b^n a^n \mid n \geq 0\}$ is not. We can prove this formally using a pumping lemma for CFLs analogous to the pumping lemma for regular sets. We will discuss the pumping lemma for CFLs in Lecture 22.

## Pushdown Automata (PDAs): A Preview

A *pushdown automaton* (PDA) is like a finite automaton, except that in addition to its finite control, it has a *stack* or *pushdown store*, which it can use to record a potentially unbounded amount of information.




The input head is read-only and may only move right. The machine can store information on the stack in a last-in-first-out (LIFO) fashion. It can *push* symbols onto the top of the stack or *pop* them off the top of the stack. It may not read down into the stack without popping the top symbols off, in which case they are lost.

We will define these machines formally in Lecture 23 and prove that the class of languages accepted by nondeterministic PDAs is exactly the class of CFLs.

## Formal Definition of CFGs and CFLs

Formally, a *context-free grammar* (CFG) is a quadruple

$$G = (N, \Sigma, P, S),$$

where

- $N$ is a finite set (the *nonterminal symbols*),
- $\Sigma$ is a finite set (the *terminal symbols*) disjoint from $N$,
- $P$ is a finite subset of $N \times (N \cup \Sigma)^*$ (the *productions*), and
- $S \in N$ (the *start symbol*).

We use capital letters $A, B, C, \ldots$ for nonterminals and $a, b, c, \ldots$ for terminal symbols. Strings in $(N \cup \Sigma)^*$ are denoted $\alpha, \beta, \gamma, \ldots$. Instead of writing productions as $(A, \alpha)$, we write $A \to \alpha$. We often use the vertical bar | as in the example above to abbreviate a set of productions with the same left-hand side. For example, instead of writing

$$A \to \alpha_1, \qquad A \to \alpha_2, \qquad A \to \alpha_3,$$

we might write

$$A \to \alpha_1 \mid \alpha_2 \mid \alpha_3.$$

If $\alpha, \beta \in (N \cup \Sigma)^*$, we say that $\beta$ *is derivable from* $\alpha$ *in one step* and write

$$\alpha \xrightarrow[G]{1} \beta$$

if $\beta$ can be obtained from $\alpha$ by replacing some occurrence of a nonterminal $A$ in $\alpha$ with $\gamma$, where $A \to \gamma$ is in $P$; that is, if there exist $\alpha_1, \alpha_2 \in (N \cup \Sigma)^*$ and production $A \to \gamma$ such that

$$\alpha = \alpha_1 A \alpha_2 \quad \text{and} \quad \beta = \alpha_1 \gamma \alpha_2.$$

Let $\xrightarrow[G]{*}$ be the reflexive transitive closure of the relation $\xrightarrow[G]{1}$; that is, define

- $\alpha \xrightarrow[G]{0} \alpha$ for any $\alpha$,
- $\alpha \xrightarrow[G]{n+1} \beta$ if there exists $\gamma$ such that $\alpha \xrightarrow[G]{n} \gamma$ and $\gamma \xrightarrow[G]{1} \beta$, and
- $\alpha \xrightarrow[G]{*} \beta$ if $\alpha \xrightarrow[G]{n} \beta$ for some $n \geq 0$.

A string in $(N \cup \Sigma)^*$ derivable from the start symbol $S$ is called a *sentential form*. A sentential form is called a *sentence* if it consists only of terminal symbols; that is, if it is in $\Sigma^*$. The *language generated by* $G$, denoted $L(G)$, is the set of all sentences:

$$L(G) = \{x \in \Sigma^* \mid S \xrightarrow[G]{*} x\}.$$

A subset $B \subseteq \Sigma^*$ is a *context-free language* (CFL) if $B = L(G)$ for some context-free grammar $G$.

Example 19.1 The nonregular set $\{a^n b^n \mid n \geq 0\}$ is a CFL. It is generated by the grammar

$$S \to aSb \mid \epsilon,$$

where $\epsilon$ is the null string. More precisely, $G = (N, \Sigma, P, S)$, where

$$N = \{S\},$$
$$\Sigma = \{a, b\},$$
$$P = \{S \to aSb,\ S \to \epsilon\}.$$

Here is a derivation of $a^3b^3$ in $G$:

$$S \xrightarrow[G]{1} aSb \xrightarrow[G]{1} aaSbb \xrightarrow[G]{1} aaaSbbb \xrightarrow[G]{1} aaabbb.$$

The first three steps apply the production $S \to aSb$ and the last applies the production $S \to \epsilon$. Thus

$$S \xrightarrow[G]{4} aaabbb.$$

One can show by induction on $n$ that

$$S \xrightarrow[G]{n+1} a^n b^n,$$

so all strings of the form $a^n b^n$ are in $L(G)$; conversely, the only strings in $L(G)$ are of the form $a^n b^n$, as can be shown by induction on the length of the derivation. □

Example 19.2 The nonregular set

$$\{\text{palindromes over } \{a,b\}^*\} = \{x \in \{a,b\}^* \mid x = \mathbf{rev}\ x\}$$

is a CFL generated by the grammar

$$S \to aSa \mid bSb \mid a \mid b \mid \epsilon.$$

The first two productions generate any number of balanced $a$'s or $b$'s at the outer ends of the string, working from the outside in. The last three productions are used to finish the derivation. The productions $S \to a$ and $S \to b$ are used to generate an odd-length palindrome with an $a$ or $b$, respectively, as the middle symbol; and the production $S \to \epsilon$ is used to generate an even-length palindrome. □

## Historical Notes

Context-free grammars and languages were introduced by Chomsky [17, 18, 20], although they were foreshadowed by the systems of Post [100] and Markov [83]. Backus–Naur form was used to specify the syntax of programming languages by Backus [7] and Naur [93].