2.0 Formal Grammars

**Context-Free Grammar (CFG).** A language generator consisting of:

1. a set T of “terminals” *(usually denoted with lowercase letters)*
2. a set N of “non-terminals” *(usually denoted with uppercase letters)*
3. a unique “start symbol” S ∈ N
4. a set P of “productions” of the form $A \rightarrow \omega$, where $A \in N$ and $\omega \in (T+N)^*$

**Notes:**

- Two rules with the same left-hand side may be combined using an OR (“|”). For example, the two rules:
  
  $A \rightarrow aBb$
  
  $A \rightarrow Aa$

  can be combined into:

  $A \rightarrow aBb | Aa$

- The language generated by the grammar G is denoted by L(G).
- Sometimes ::= is used in place of $\rightarrow$. This is called Backus-Naur Form (BNF).
- A grammar describes the syntax rules for forming sentences. It tells us whether a particular string is “well-formed”, or “legal”.
- No satisfactory grammar has yet been developed for natural languages. Natural languages are inherently ambiguous and context-sensitive.

**An Example inspired by the English language:**

<table>
<thead>
<tr>
<th>Sentence Structure</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;sentence&gt;</td>
<td>$\rightarrow$ &lt;noun-phrase&gt; &lt;predicate&gt;</td>
</tr>
<tr>
<td>&lt;noun-phrase&gt;</td>
<td>$\rightarrow$ &lt;article&gt; &lt;noun&gt;</td>
</tr>
<tr>
<td>&lt;predicate&gt;</td>
<td>$\rightarrow$ &lt;verb&gt;</td>
</tr>
<tr>
<td>&lt;article&gt;</td>
<td>$\rightarrow$ the</td>
</tr>
<tr>
<td>&lt;noun&gt;</td>
<td>$\rightarrow$ cat</td>
</tr>
<tr>
<td>&lt;verb&gt;</td>
<td>$\rightarrow$ jumps</td>
</tr>
</tbody>
</table>

“the cat jumps” and “a flower blooms” would be legal in the above grammar.

A grammar only defines what is syntactically correct, not necessarily what is meaningful. For example the above grammar will also generate sentences such as “the cat blooms”, and “a flower jumps” which of course do not make sense.
A grammar describes the syntax (the form) and not the semantics (the meaning). However, the syntax may, indirectly, affect the semantics (more on this later).

**Two ways that grammars are used in compiling computer programs:**

1. Given a grammar $G$ and a string $\omega$, determine if $\omega \in L(G)$. This is “parsing.”
2. Given a description of language $L$, design a grammar $G$ that generates $L$.

**Example:**

Consider the grammar $G_1$ with the following rules:

- $S \rightarrow aA$
- $S \rightarrow AS$
- $A \rightarrow b$
- $A \rightarrow \lambda$

Is the string $ba \in L(G_1)$? That is, is $ba$ “legal” according to grammar $G$?

**Answer:** The string $ba$ is legal only if it can be derived using $G$.

Start with $S$. Apply one rule at a time, replacing a nonterminal with the right side of an applicable rule, until only terminals remain.

A derivation of $ba$ using grammar $G$ is:

- $S \Rightarrow AS \Rightarrow AaA \Rightarrow baA \Rightarrow ba$

**Example:**

Build a grammar for all strings (of $a$'s & $b$'s) that begin with $a$ and end with $b$.

**Solution 1:**

- $S \rightarrow aAb$
- $A \rightarrow aA | bA | \lambda$

**Solution 2:** (FSA style: generate one character at a time from left to right)

- $S \rightarrow aA$ // start with an $a$
- $A \rightarrow aA | bA | B$ // produce either $a$ or $b$, repeatedly
- $B \rightarrow b$ // produce a final $b$ and quit