Grammars play a crucial role in programming languages because they precisely capture the syntactic nature of programming languages.

We start our discussion of grammars by looking at the nature of sequences of symbols, where sequences of symbols form the foundation of any language, both natural and artificial.

We will call sequences of symbols *strings*. 
Strings

**Definition:** [Strings over an Alphabet]¹

- An *alphabet* is a finite, nonempty set – we refer to the elements of an alphabet as *symbols*.

- A finite sequence of symbols from a given alphabet is called a *string over the alphabet*.

- A string that consists of a sequence $a_1, a_2, \ldots, a_n$ of symbols is denoted by the juxtaposition $a_1 a_2 \ldots a_n$.

- The length of some string $s$ is denoted by $|s|$ and assumed to equal the number of symbols in the string.

- Strings that have zero symbols, called *empty strings*, are denoted by $\epsilon$ with $|\epsilon| = 0$.

---

Example: Let $\Gamma_1 = \{a, \ldots, z\}$ and $\Gamma_2 = \{0, \ldots, 9\}$ is alphabets. $abb$ is a string over $\Gamma_1$, and $123$ is a string over $\Gamma_2$. $ba12$ is neither a string over $\Gamma_1$ nor a string over $\Gamma_2$ but it is a string over $\Gamma_1 \cup \Gamma_2$. The string $314\ldots$ is not a string over $\Gamma_2$, because it is not a finite sequence.

Some other observations:

- The empty string $\epsilon$ is a string over any alphabet.
- The empty set $\emptyset$ is not an alphabet because it contains no element.
- The set of natural numbers is not an alphabet, because it is not finite.
Definition: [Kleene Closure] Given some alphabet \( \Gamma \) then the set of all possible strings over \( \Gamma \) including the empty string \( \epsilon \) is denoted by \( \Gamma^* \) and is called the Kleene Closure of \( \Gamma \). (Similar to the power set construction with the exception that the constructed set is always infinite.)

Example: Let \( \Gamma = \{ a \} \), then \( \Gamma^* = \{ \epsilon, a, aa, aaa, aaaa, \ldots \} \).

Example: Let \( \Gamma = \{ a, b \} \), then

\[
\Gamma^* = \{ \epsilon, a, b, aa, bb, ab, ba, aaa, aab, \ldots \}.
\]
**Definition:** [String Concatenation] Given some alphabet $\Gamma$ with $s_1 \in \Gamma^*$ and $s_2 \in \Gamma^*$, then the *concatenation of the strings* written as $s_1 s_2$ also belongs to $\Gamma^*$, that is the string $s_1 s_2 \in \Gamma^*$. 
**Definition:** [Context-Free Grammar] A *context-free grammar* is a triple $(\Gamma, \rightarrow, \gamma)$ such that,

- $\Gamma = T \cup N$ with $T \cap N = \emptyset$, where $T$ is a set of symbols called the *terminals* and $N$ is a set of symbols called the *non-terminals*,\(^2\)
- $\rightarrow \subseteq N \times \Gamma^*$ is a set of rules of the form $u \rightarrow v$ with $u \in N$ and $v \in \Gamma^*$,
- $\gamma$ is called the *start symbol* and $\gamma \in N$.

\(^2\)The fact that $T$ and $N$ are non-overlapping means that there will never be confusion between terminals and non-terminals.
Example: Grammar for arithmetic expressions. We define the grammar $(\Gamma, \to, \gamma)$ as follows:

- $\Gamma = T \cup N$ with $T = \{a, b, c, +, *, (, )\}$ and $N = \{E\}$,
- $\to$ is is defined as,

\[
\begin{align*}
E & \to E + E \\
E & \to E \ast E \\
E & \to (E) \\
E & \to a \\
E & \to b \\
E & \to c
\end{align*}
\]

- $\gamma = E$ (clearly this satisfies $\gamma \in N$).
In order for a grammar $(\Gamma, \rightarrow, \gamma)$ to be useful we allow rules to be applied to *substrings* of given strings; let $s = xuy, t = xvy$ with $x, y, v \in \Gamma^*$, $u \in N$, and a rule $u \rightarrow v$, then we say that $s$ *rewrites to* $t$ and we write,

$$s \Rightarrow t.$$ 

More formally,

**Definition:** [one-step rewriting relation] Let $(\Gamma, \rightarrow, \gamma)$ be a be context-free grammar, then the *one-step rewriting relation* $\Rightarrow \subseteq \Gamma^* \times \Gamma^*$ is the set with $(s, t) \in \Rightarrow$ for strings $s, t \in \Gamma^*$ if and only if there exist $x, y, v \in \Gamma^*$ and $u \in N$ with $s = xuy, t = xvy$, and $u \rightarrow v$.

In plain English: any two strings $s, t$ belong to the relation $\Rightarrow$ if and only if they can be related by a rewrite rule.
With grammars, derivations always start with the start symbol \( \gamma \in \Gamma^* \). Consider,

\[
E \Rightarrow E*E \Rightarrow (E)*E \Rightarrow (E+E)*E \Rightarrow (a+E)*E \Rightarrow (a+b)*E \Rightarrow (a+b)*c.
\]

Here, \((a+b)*c\) is a *normal form* often also called a *terminal- or derived-string*. Normal forms are characterized by the fact that no additional rewriting can be applied to them.

We often write,

\[
E \Rightarrow^* (a + b) * c
\]

stating that the normal form is derivable from the start symbol in zero or more steps.
**Exercise:** Identify the rule that was applied at each rewrite step in the above derivation.

**Exercise:** Derive the string \((a)\).

**Exercise:** Derive the string \(a + b \times c\).
We are now in the position to define exactly what we mean by a *language*.

**Definition:** [Language] Let $G = (\Gamma, \rightarrow, \gamma)$ be a context-free grammar, then we define the *language of grammar* $G$ as the set of all terminal strings that can be derived from the start symbol $\gamma$ by rewriting using the rules in $\rightarrow$. Formally,

$$L(G) = \{ q \mid \gamma \Rightarrow^* q \land q \in T^* \}.$$

**Example:** Let $J = (\Gamma, \rightarrow, \gamma)$ be the grammar of Java, then $L(J)$ is the set of all possible Java programs. The Java language is defined as the set of all possible Java programs.

---

*Observe that $T^*$ is the set of all terminal strings.*
Observations:

- With the concept of a language we can now ask interesting questions. For example, given a grammar $G$ and some sentence $p \in T^*$, does $p$ belong to $L(G)$?

- If we let $J$ be the grammar of Java, then asking whether some string $p \in T^*$ is in $L(J)$ is equivalent to asking whether $p$ is a **syntactically correct program**.
**Example:** A simple imperative language. We define grammar $G = (\Gamma, \rightarrow, \gamma)$ as follows:

- $\Gamma = T \cup N$ where
  
  $$T = \{0, \ldots, 9, a, \ldots, z, \text{true}, \text{false}, \text{skip}, \text{if}, \text{then}, \text{else}, \text{while}, \text{do}, \text{end}, +, -, *, =, \leq, !, &&, ||, :=, ;, (, )\}$$

  and

  $$N = \{A, B, C, D, L, V\}.$$

- The rule set $\rightarrow$ is defined by,

  \[
  \begin{align*}
  A & \rightarrow \ D | V | A + A | A - A | A \ast A | (A) \\
  B & \rightarrow \ \text{true} | \text{false} | A = A | A \leq A | !B | B\&\&B | B||B | (B) \\
  C & \rightarrow \ \text{skip} | V := A | C ; C | \text{if} B \text{ then } C \text{ else } C \text{ end} | \text{while} B \text{ do } C \text{ end} \\
  D & \rightarrow \ L | - L \\
  L & \rightarrow \ 0 L | \ldots | 9 L | 0 | \ldots | 9 \\
  V & \rightarrow \ a V | \ldots | z V | a | \ldots z
  \end{align*}
  \]

- $\gamma = C$. 

\[\]
Here are some elements in \( L(G) \),

\[
\begin{align*}
x & := 1; \quad y := x \\
v & := 1; \quad \textbf{if} \ v \leq 0 \ \textbf{then} \ v := (-1) \times v \ \textbf{else} \ \textbf{skip} \ \textbf{end} \\
n & := 5; \quad f := 1; \quad \textbf{while} \ 2 \leq n \ \textbf{do} \ f := n \times f; \ n := n - 1 \ \textbf{end}
\end{align*}
\]

**Exercise:** Prove that they belong to \( L(G) \).
Reading: Denotational Semantics/Schmidt – pages 5 thru 8.
Assignment #1 – see BrightSpace