2.8 On limitations of regular languages
For cardinality reasons there must be (many) non-regular languages: languages are uncountable, but regular expressions are countable.
Could we find a concrete, interesting example of a non-regular language? Easily.
The basic limitation of regular languages: finite state automata only have a limited “memory”. Thus they cannot solve problems that require remembering arbitrarily large numbers.
Example: the parenthesis language

\[ L_{\text{match}} = \{ (k)^k \mid k \geq 0 \}. \]

Formalization: “the pumping lemma”.

---

Lemma 2.6 (The pumping lemma) Let \( A \) be a regular language. Then there exists an \( n \geq 1 \), such that every \( x \in A \), \( |x| \geq n \), can be (somehow) split into parts \( x = uvw \) such that \( |uv| \leq n \), \( |v| \geq 1 \), and \( uv^i w \in A \) for all \( i \in \{0, 1, 2, \ldots \} \).

Proof. Let \( M \) be a deterministic finite state automaton that recognizes \( A \), and let \( n \) be the number of states in \( M \). Examine the sequence of states \( M \) goes through with input \( x \in A \), \( |x| \geq n \). Since \( M \) moves from one state to another with each input symbol, it must visit one state (at least) twice—in fact, already within the first \( n \) symbols of \( x \). Let \( q \) be the first state visited twice.

Let \( u \) be the prefix of \( x \) the machine \( M \) has handled when it first enters \( q \), \( v \) the part of \( x \) handled before its first return to \( q \) and \( w \) the remaining part of \( x \). Then we have \( |uv| \leq n \), \( |v| \geq 1 \), \( uv^i w \in A \) for all \( i \in \{0, 1, 2, \ldots \} \).

---

3. Grammars and production of strings
A grammar = a transformation system for producing strings (“words” of a language) from a certain initial string by repeatedly rewriting substrings according to given rules.
A grammar is context-free if in each rewriting step one so-called nonterminal symbol may be replaced by some substituted string, and the substitution can always be carried out without regard for the surrounding structure of the string.
Applications: describing structured texts (e.g., BNF-syntax descriptions of programming languages, DTD/Schema definitions of XML), more generally describing structured “objects” (e.g., syntactic pattern recognition).
Context-free grammars can also describe (produce) certain non-regular languages.

*Example:* A context-free grammar for the language \( l_{\text{match}} \) (initial symbol \( S \)):

(i) \( S \rightarrow \varepsilon \)
(ii) \( S \rightarrow (S) \)

E.g., the derivation of the string (((()))):

\[
S \Rightarrow (S) \Rightarrow ((S)) \Rightarrow (((S))) \Rightarrow (((\varepsilon))) = (((()))).
\]

---

Another example: a simplified grammar for arithmetic expressions of a programming language similar to C:

\[
\begin{align*}
E & \rightarrow T \quad | \quad E + T \\
T & \rightarrow F \quad | \quad T * F \\
F & \rightarrow a \quad | \quad (E).
\end{align*}
\]

E.g., production of the expression \((a + a) * a\):

\[
\begin{align*}
E & \Rightarrow I \\
& \Rightarrow (E) * F \\
& \Rightarrow (E + T) * F \\
& \Rightarrow (I + T) * F \\
& \Rightarrow (a + I) * F \\
& \Rightarrow (a + E) * F \\
& \Rightarrow (a + a) * E \\
& \Rightarrow (a + a) * a.
\end{align*}
\]

---

**Definition 3.1** A context-free grammar is a 4-tuple

\[
G = (V, \Sigma, P, S),
\]

where

- \( V \) is the alphabet of the grammar;
- \( \Sigma \subseteq V \) is the set of terminals in the grammar; its complement
  \( N = V - \Sigma \) is the set of nonterminals;
- \( P \subseteq N \times V^* \) is the set of rules or productions;
- \( S \in N \) is the initial symbol.

The production \((A, \omega) \in P\) is usually denoted by \( A \rightarrow \omega \).

The string \( \gamma \in V^* \) produces or derives directly the string \( \gamma' \in V^* \) in
grammars \( G \), denoted by

\[
\gamma \xrightarrow{G} \gamma'
\]

if we may write \( \gamma = \alpha A \beta, \gamma' = \alpha \omega \beta \) \((\alpha, \beta, \omega \in V^*, A \in N)\), and \( G \)
contains the production \( A \rightarrow \omega \).

If the grammar \( G \) is clear from the context, we may write \( \gamma \xrightarrow{} \gamma' \).

The string \( \gamma \in V^* \) produces or derives the string \( \gamma' \in V^* \) in grammar
\( G \), denoted by

\[
\gamma \xrightarrow{G} \gamma'
\]

if there is a sequence \( \gamma_0, \gamma_1, \ldots, \gamma_n \) \((n \geq 0)\) of strings over \( V \) such that

\[
\gamma = \gamma_0 \xrightarrow{G} \gamma_1 \xrightarrow{G} \cdots \xrightarrow{G} \gamma_n = \gamma'.
\]

As a special case \( n = 0 \) we have \( \gamma \xrightarrow{} \gamma' \) for all \( \gamma \in V^* \). Again, if \( G \) is
obvious from the context, we may write \( \gamma \xrightarrow{G} \gamma' \).
A string $\gamma \in V^*$ is a sentence derivation of $G$, if $S \Rightarrow^* \gamma$.
A sentence derivation $x \in \Sigma^*$ of $G$ that only contains terminal symbols is a sentence of $G$.
The language produced or described by $G$ consists of the sentences of $G$:
$$L(G) = \{ x \in \Sigma^* \mid S \Rightarrow^* x \}.$$  
A formal language $L \subseteq \Sigma^*$ is context-free, if it can be produced by a context-free grammar.

---

For example the parenthesis language $L_{\text{match}} = \{(k)^k \mid k \geq 0\}$ is produced by the grammar
$$G_{\text{match}} = (\{S, (, )\}, \{(, ), (,)\}, S \rightarrow \varepsilon, S \rightarrow (S), S).$$

The language $L_{\text{expr}}$ of simple arithmetic expressions is produced by the grammar
$$G_{\text{expr}} = (V, \Sigma, P, E),$$
where
$$V = \{E, T, F, a, +, *, (, )\},$$
$$\Sigma = \{a, +, *, (, )\},$$
$$P = \{E \rightarrow T, E \rightarrow E + T, E \rightarrow F, T \rightarrow F, T \rightarrow T * F, E \rightarrow a, F \rightarrow (E)\}.$$  

Another grammar for producing $L_{\text{expr}}$ is
$$G'_{\text{expr}} = (V, \Sigma, P, E),$$
where
$$V = \{E, a, +, *, (, )\},$$
$$\Sigma = \{a, +, *, (, )\},$$
$$P = \{E \rightarrow E + E, E \rightarrow E * E, E \rightarrow a, E \rightarrow (E)\}.$$  

Note: Although $G'_{\text{expr}}$ appears simpler than the grammar $G_{\text{expr}}$, it is unfortunately structurally ambiguous, which is often an undesirable property.

---

Established notation
Nonterminals: $A, B, C, \ldots, S, T$.
Terminals: letters $a, b, c, \ldots, s, t$;
numbers $0, 1, \ldots, 9$;
special signs; bolded or underlined reserved words (if, for, end, ...).
Arbitrary symbols (when no distinction is made between terminals and nonterminals): $X, Y, Z$.
Strings of terminals: $u, v, w, x, y, z$.
Mixed strings: $a, b, \gamma, \ldots, \omega$.  

Productions with a common left side $A$ may be written together:

instead of

$$A \rightarrow \omega_1, \ A \rightarrow \omega_2, \ldots A \rightarrow \omega_k$$

we write

$$A \rightarrow \omega_1 | \omega_2 | \ldots | \omega_k.$$  

A grammar is often presented simply as a collection of rules

$$A_1 \rightarrow \omega_{11} | \ldots | \omega_{1k_1}$$
$$A_2 \rightarrow \omega_{21} | \ldots | \omega_{2k_2}$$
$$\vdots$$
$$A_m \rightarrow \omega_{m1} | \ldots | \omega_{mk_m}.$$

Then nonterminals are determined by the notation conventions above or by observing that they appear on the left side of productions; other symbols are terminals. The initial symbol is the nonterminal that appears as the left side of the first rule; in this case $A_1$.

Some constructions

Let $L(T)$ be the set of strings that can be derived from the nonterminal $T$. Let us have a collection $P$ of productions, where the nonterminal $A$ does not appear and for which $L(B)$ can be derived from $B$ and similarly $L(C)$ from $C$.

By adding one of the following productions to $P$ we obtain new languages:

<table>
<thead>
<tr>
<th>production</th>
<th>language</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \rightarrow B</td>
<td>C$</td>
</tr>
<tr>
<td>$A \rightarrow BC$</td>
<td>concatenation $L(A) = L(B)L(C)$, and</td>
</tr>
<tr>
<td>$A \rightarrow AB</td>
<td>\epsilon$ (left recursion) or $A \rightarrow BA</td>
</tr>
</tbody>
</table>

3.2 Regular languages and context-free grammars

We have seen that context-free grammars can describe some non-regular languages such as $L_{\text{match}}$ and $L_{\text{expr}}$. We show that all regular languages can also be described by context-free grammars. Thus context-free languages are a proper superclass of regular languages.

A context-free grammar is right linear, if all its productions are of the form $A \rightarrow aB$ or $A \rightarrow \epsilon$, and left linear, if all its productions are of the form $A \rightarrow Ba$ or $A \rightarrow \epsilon$.

It turns out that all regular languages and no others can be produced by left or right linear grammars. Therefore such grammar are also called regular. We prove this only for right linear grammars.
Theorem 3.1 Every regular language can be produced by a right linear grammar.

Proof. Let $L$ be a regular language over the alphabet $\Sigma$ and let $M = (Q, \Sigma, \delta, q_0, F)$ be the (deterministic or nondeterministic) finite state automaton that recognizes $L$. We design a grammar $G_M$, for which $L(G_M) = L(M) = L$.

The set of terminals of $G_M$ is the same as the input alphabet of $M$, and the set of nonterminals will have one symbol $A_q$ for each state $q$ in $M$. The initial symbol of the grammar is $A_{q_0}$, and its productions represent the transitions of $M$:

(i) for each final state $q \in F$ of $M$ the production $A_q \rightarrow \varepsilon$ is included;
(ii) for each transition $q \xrightarrow{\alpha} q'$ in $M$ (that is $q' \in \delta(q, \alpha)$) the production $A_q \rightarrow aA_{q'}$ is included.

To verify the correctness of the construction we denote the set of strings of terminals that can be derived from $A_q$ by

$L(A_q) = \{ x \in \Sigma^* | A_q \Rightarrow^* x \}$

By induction over the length of $x$ we may show that for all $q$ we have

$x \in L(A_q) \iff (q, x) \xrightarrow{\alpha} (q_r, \varepsilon)$ for some $q_r \in F$.

In particular we have

$L(G_M) = L(A_{q_0}) = \{ x \in \Sigma^* | (q_0, x) \xrightarrow{\alpha} (q_r, \varepsilon) \hspace{1cm}$

for some $q_r \in F \}

= L(M) = L \hspace{1cm} \Box$

Example. Automaton:

![Diagram of a simple automaton with states 1 and 2, transitions a and b]

The corresponding grammar:

$A_1 \rightarrow aA_1 | bA_1 | bA_2$

$A_2 \rightarrow \varepsilon | bA_2$

Theorem 3.2 Every language produced by a right linear grammar is regular.

Proof. Let $G = (V, \Sigma, P, S)$ be a right linear grammar. Construct a nondeterministic finite state automaton $M_G = (Q, \Sigma, \delta, q_0, F)$ that recognizes $L(G)$ as follows:

The states of $M_G$ represent the nonterminals of $G$: $Q = \{ q_A | A \in V - \Sigma \}$

The initial state of $M_G$ is the state $q_S$ that represents the initial symbol $S$.

The input alphabet of $M_G$ is the set $\Sigma$ of terminals of $G$.

The transition function $\delta$ of $M_G$ emulates the productions of $G$ so that for every production $A \rightarrow aB$ the automaton has the transition $q_A \xrightarrow{a} q_B$ (i.e., $q_B \in \delta(q_A,a)$).
The final states of $M_G$ are those states whose corresponding nonterminals in $G$ have an $\varepsilon$-production:

$$F = \{ q_A \in Q \mid A \rightarrow \varepsilon \in P \}.$$ 

The correctness of the construction can again be verified by induction on the length of strings produced by $G$ and accepted by $M_G$. \qed