1.4 Nonregular Languages

- The number of formal languages over any alphabet (= decision/recognition problems) is uncountable
- On the other hand, the number of regular expressions (= strings) is countable
- Hence, all languages cannot be regular
- Can we find an intuitive example of a nonregular language?
- The language of balanced pairs of parentheses

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

**Theorem 1.70 (Pumping lemma)**

Let \( A \) be a regular language. Then there exists \( p \geq 1 \) (the pumping length) s.t. any string \( s \in A, |s| \geq p \), may be divided into three pieces, \( s = xyz \), satisfying the following conditions:

- \(|xy| \leq p\),
- \(|y| \geq 1\) and
- \(xy^iz \in A \quad \forall i = 0, 1, 2, ...

**Proof.** Let \( M = (Q, \Sigma, \delta, q_0, F) \) be a DFA that recognizes \( A \) s.t. \(|Q| = p\). When the DFA is computing with input \( s \in A, |s| \geq p \), it must pass through some state at least twice when processing the first \( p \) characters of \( s \). Let \( q \) be the first such state.
Let us choose so that:

- $x$ is the prefix of $s$ that has been processed when $M$ enters $q$ for the first time,
- $y$ is that part of the suffix $s$ that gets processed by $M$ before it re-enters state $q$, and
- $z$ is the rest of the string $s$.

Obviously $|xy| \leq p$, $|y| \geq 1$ and $xy^iz \in A$ for all $i = 0, 1, 2, \ldots$

\[ \begin{array}{c}
\text{Observe:} \text{ The pumping lemma does not give us liberty to choose } x \text{ and } y \text{ as we please.}
\end{array} \]

**Example**

Let us assume that $L_{\text{parenth}}$ is a regular language.

By the pumping lemma there exists some number $p$ s.t. strings of $L_{\text{parenth}}$ of length at least $p$ can be pumped. Let us choose $s = (p)^p$. Then $|s| = 2p > p$.

By Lemma 1.70 $s$ can be divided into three parts $s = xyz$ s.t. $|xy| \leq p$ and $|y| \geq 1$. Therefore, it must be that

- $x = (i \ i \leq p-1$,
- $y = (j \ j \geq 1$, and
- $z = (p-(i+j))p$.

By our assumption $xy^iz \in L_{\text{parenth}}$, for all $k = 0, 1, 2, \ldots$, but for example $xy^0z = xz = (i (p+i))p = (p+j)p \not\in L_{\text{parenth}}$.

Because $p+j \neq p$ since $j \geq 1$.

Hence, $L_{\text{parenth}}$ cannot be a regular language.
The main limitation that finite automata have is that they have no (external) means of keeping track of an unlimited number of possibilities; i.e., to count.

Consider the following two languages:

\[ C = \{ w \mid w \text{ has an equal number of } 0\text{s and } 1\text{s} \} \]

\[ D = \{ w \mid w \text{ has an equal number of occurrences of } 01 \text{ and } 10 \text{ as substrings} \} \]

At first glance, a recognizing machine needs to count in each case.

The language \( C \) contains \( \{ 0^k 1^k \mid k \geq 0 \} \) as a subset and, hence, the nonregularity of \( L_{\text{parenth}} \) proves that of \( C \).

Surprisingly, \( D \) is regular.
Example 1.75

Let $F = \{ ww \mid w \in \{ 0, 1 \}^* \}$. We show that $F$ is not regular.

Assume that $F$ is regular. Let $p$ be the pumping length given by the pumping lemma. Let $s$ be the string $0^p10^p1$. Because $s$ is a member of $F$ and it has length more than $p$, the pumping lemma guarantees that $s$ can be split into pieces $s = xyz$, satisfying the three conditions of the lemma. We show that this outcome is impossible.

Because $|xy| \leq p$, $x$ and $y$ must consist only of 0s, so $xyyz \notin F$.

More exactly, $x = 0^i$, $y = 0^j$, and $z = 0^{p-(i+j)}10^p1$.

Therefore, $xy^2z = xyyz = 0^{i+j}10^p1 = 0^{p+1}10^p1$ which does not belong to $F$ since $0^{p+1}1$ has more zeros than $0^p1$ since by pumping lemma $j \geq 1$. Hence, $F$ is not a regular language.

Example 1.77

Let $E = \{ 0^i1^j \mid i > j \}$. We show that $E$ is not regular.

Assume that $E$ is regular. Let $p$ be the pumping length for $E$ given by the pumping lemma. Let $s$ be the string $0^p10^p1$. Then $s$ can be split into $xyz$ satisfying the conditions of the pumping lemma.

Because $|xy| \leq p$, $x$ and $y$ must consist only of 0s: $x = 0^i$ and $y = 0^j$.

Let us examine the string $xyyz$ to see whether it can be in $E$.

Adding an extra copy of $y$ increases the number of 0s. But $E$ contains all strings in $0^*1^*$ that have more 0s than 1s, so increasing the number of 0s will still give a string in $E$.

We need to pump down: $xy^0z = xz = 0^{p+1-j}1^p = 0^{p+1}1 \notin E$ since $p+1-j \leq p$ because by assumption $j \geq 1$. Hence, the claim follows.
2. Context-Free Languages

- The language of balanced pairs of parentheses is not a regular one
- On the other hand, it can be described using the following substitution rules
  1. \( S \rightarrow \varepsilon \) and
  2. \( S \rightarrow (S) \)

- These productions generate the strings of the language \( L_{\text{parenth}} \) starting from the start variable \( S \)
  \[
  S \Rightarrow (S) \Rightarrow (((S))) \Rightarrow (((((S)))) = (((())))
  \]

- The string being described is generated by substituting variables one by one according to the given rules
- The string surrounding a variable does not determine the chosen production \( \Rightarrow \) context-free grammar
- One often abbreviates
  \[
  A \rightarrow w_1 \mid \ldots \mid w_k
  \]
  to describe the alternative productions associated with the variable \( A \)
  \[
  A \rightarrow w_1, \ldots, A \rightarrow w_k
  \]
- \( S \rightarrow \varepsilon \mid (S) \)
Simple arithmetic expressions

(E = expression, T = term and F = factor)

\[ E \rightarrow E + T \mid T \]
\[ T \rightarrow T \times F \mid F \]
\[ F \rightarrow (E) \mid a \]

Generation the expression \((a + (a)) \times a\)

\[ E \Rightarrow T \Rightarrow T \times F \Rightarrow F \times F \Rightarrow (E) \times F \Rightarrow (E + T) \times F \Rightarrow (T + T) \times F \Rightarrow (F + T) \times F \Rightarrow (a + T) \times F \Rightarrow (a + F) \times F \Rightarrow (a + (E)) \times F \Rightarrow (a + (T)) \times F \Rightarrow (a + (F)) \times F \Rightarrow (a + (a)) \times F \Rightarrow (a + (a)) \times a \]

**Definition 2.2** A context-free grammar is a 4-tuple \( G = (V, \Sigma, R, S) \), where
- \( V \) is a finite set called the **variables**, 
- \( \Sigma \) is a finite set, disjoint from \( V \), called the **terminals**
- \( V \cup \Sigma \) is the **alphabet** of \( G \),
- \( R \subseteq V \times (V \cup \Sigma)^* \) is a finite set of **rules**, and
- \( S \in V \) is the **start variable**

\((A, w) \in R\) is usually denoted as \( A \rightarrow w \)
Let $G = (V, \Sigma, R, S)$, strings $u, v, w \in (V \cup \Sigma)^*$, and $A \rightarrow w$ a production in $R$

- $uAv$ yields string $uvw$ in grammar $G$, written $uAv \Rightarrow_G uvw$
- String $u$ derives string $v$ in grammar $G$, written $u \Rightarrow_G v$,
  if a sequence $u_1, u_2, \ldots, u_k \in (V \cup \Sigma)^*$ ($k \geq 0$) exists s.t.
  $u \Rightarrow_G u_1 \Rightarrow_G u_2 \Rightarrow_G \ldots \Rightarrow_G u_k \Rightarrow_G v$
- $k = 0$: $u \Rightarrow_G u$ for any $u \in (V \cup \Sigma)^*$

$u \in (V \cup \Sigma)^*$ is a sentential form of $G$ if

$S \Rightarrow_G u$

A sentential form consisting of only terminals $w \in \Sigma^*$ is a sentence of $G$

- The language of the grammar $G$ consists of sentences
  $L(G) = \{ w \in \Sigma^* | S \Rightarrow_G w \}$

A formal language $L \subseteq \Sigma^*$ is context-free, if it can be generated using a context-free grammar
A context-free grammar is **right-linear** if all its productions are of type $A \rightarrow \epsilon$ or $A \rightarrow aB$

**Theorem** Any regular language can be generated using a right-linear context-free grammar.

**Theorem** Any right-linear context-free language is regular.

- Hence, right-linear grammars generate exactly regular languages
- However, there are context-free languages which are not regular; e.g., the language of balanced pairs of parentheses $L_{paren}$
- Therefore, context-free languages are a proper superset of regular languages

**Ambiguity**

- The sequence of one-step derivations leading from the start variable $S$ to string $w$

\[ S \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_k \Rightarrow w \]

is called the derivation of $w$

In the grammar for arithmetic expressions the sentence $a+a$ can be derived in many different ways:

1. $E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + F \Rightarrow a + a$
2. $E \Rightarrow E + T \Rightarrow E + F \Rightarrow T + F \Rightarrow F + F \Rightarrow F + a \Rightarrow a + a$
3. $E \Rightarrow E + T \Rightarrow E + F \Rightarrow E + a \Rightarrow T + a \Rightarrow F + a \Rightarrow a + a$

- The differences caused by varying substitution order of variables can be abstracted away by examining parse trees
Context-free grammar $G$ is *ambiguous* if some sentence of $G$ has two (or more) distinct parse trees

- Otherwise the grammar is *unambiguous*

- Language that has no unambiguous context-free grammar is *inherently ambiguous*

- E.g. language $\{ a^i b^j c^k \mid i = j \lor j = k \}$ is inherently ambiguous

- An alternative grammar for the simple arithmetic expressions:

  $$ E \rightarrow E + E \mid E \times E \mid (E) \mid a $$
Chomsky Normal Form

Definition 2.8 A context-free grammar is in Chomsky normal form (CNF), if
• At most the start variable $S$ derives the empty string,
• Every rule is of the form $A \rightarrow BC$ or $A \rightarrow a$
  (except maybe $S \rightarrow \epsilon$),
• The start variable $S$ does not appear in the right-hand side of any rule.

Theorem 2.9 Any context-free language is generated by a context-free grammar in CNF.

Proof We convert any grammar into CNF. The conversion has three stages. First, we add a new start variable. Then, we eliminate all $\epsilon$ rules and unit rules.
Eliminating $\varepsilon$ rules

**Lemma** Any context-free language can be converted into an equivalent grammar in which at most the start variable derives the empty string.

**Proof**
Let $G = (V, \Sigma, R, S)$.
Computing the variables of $G$ that derive the empty string:

1. $\text{NULL} = \{ A \in V | A \rightarrow \varepsilon \in R \}$
2. Repeat until set $\text{NULL}$ does not change any more:
   
   $$\text{NULL} += \{ A \in V | A \rightarrow B_1 \ldots B_k \in R, B_i \in \text{NULL} \forall i = 1, \ldots, k \}$$

Each rule $A \rightarrow X_1 \ldots X_k$ in $G$ is replaced by the set of all such rules that are of form $A \rightarrow \alpha_1 \ldots \alpha_k$, where

$$\alpha_i = \begin{cases} X_i & \text{if } X_i \not\in \text{NULL} \\ X_i | \varepsilon & \text{if } X_i \in \text{NULL} \end{cases}$$

In the end we remove all rules that have the form $A \rightarrow \varepsilon$.

If $S \rightarrow \varepsilon$ belongs to the removed rules, we take a new start variable $S'$ for the grammar and give it rules $S' \rightarrow S | \varepsilon$. 
Eliminating unit rules

A unit rule has the form $A \to B$, where $A$ and $B$ are variables.

**Lemma.** Any context-free language can be converted into an equivalent grammar which has no unit rules.

**Proof.** Let $G = (V, \Sigma, R, S)$. Computing the unit followers for each variable in $G$:

1. $F(A) = \{ B \in V \mid A \to B \in R \}$
2. Until the $F$-sets do not change anymore
   
   $F(A) += \{ F(B) \mid A \to B \in R \}$

In the end we remove all unit rules in $G$ and replace them by all rules of the form $A \to \Omega$, where $B \in F(A)$ and $B \to \Omega$. $\Box$
Once all ε rules and unit rules have been eliminated, all rules have form $A \rightarrow a$, $A \rightarrow X_1 \ldots X_k$, $k \geq 2$, or $S \rightarrow ε$.

For every $a \in \Sigma$ we add to the grammar the variable $C_a$ and rule $C_a \rightarrow a$.

A rule $A \rightarrow X_1 \ldots X_k$, $k \geq 2$, is replaced by a set of rules

\[
A \rightarrow X'_1 A_1 \\
A_1 \rightarrow X'_2 A_2 \\
\vdots \\
A_{k-2} \rightarrow X'_{k-2} A_{k-1} \\
A_{k-1} \rightarrow X'_{k-1} X'_{k}
\]

where

\[
X'_i = \begin{cases} 
X_i & \text{if } X_i \in V \\
C_a & \text{if } X_i = a \in \Sigma 
\end{cases}
\]
Algorithm CYK

- The strings of a context-free grammar that has been converted into CNF can be parsed in $\Theta(n^3)$ time using the Cocke-Younger-Kasami algorithm
- In other words, context-free languages can be efficiently recognized
- The operating principle of algorithm CYK is dynamic programming
- For each substring we tabulate those variables from which the substring can be derived from
- If in the end the start variable of the grammar belongs to the set of variables that derive the whole string, the string at hand belongs to the language
2.2 Pushdown Automata

- Pushdown automata are like NFAs, but have an extra component: (an infinite) stack
- We can write a new symbol on the stack at the top by pushing it
- We can read and remove the top symbol from the stack by popping it
- In a pushdown automaton the transitions always also concern the stack
- The stack gives the automaton a "memory" by which we can avoid some of the limitations that finite automata have

Definition 2.13

A pushdown automaton is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where
- $Q$ is the finite set of states,
- $\Sigma$ is the input alphabet,
- $\Gamma$ is the stack alphabet,
- $q_0 \in Q$ is the start state,
- $F \subseteq Q$ is the set of accept states, and
- $\delta$ is the set-valued transition function:
  $$\delta : Q \times \Sigma \times \Gamma_{\varepsilon} \rightarrow P(Q \times \Gamma_{\varepsilon})$$
In general pushdown automata are nondeterministic:
\[ \delta(r, x, a) = \{ (r_1, b_1), \ldots, (r_k, b_k) \} \]

- By reading the input symbol \( x \) and stack symbol \( a \)
- The automaton may transfer from state \( r \) to one of the states \( r_1, \ldots, r_k \), and
- Simultaneously replace the top symbol of the stack by one of the symbols \( b_1, \ldots, b_k \).

1. If \( x = \epsilon \), the automaton transfers without reading an input symbol;
2. If \( a = \epsilon \), the automaton does not read a stack symbol, but writes a new symbol at the top of the stack, leaving the old top symbol as is (push);
3. If \( a \neq \epsilon \) and \( b_i = \epsilon \), the top symbol of the stack is read and removed, but no new symbol is not written in its stead (pop)