1.3 Regular Expressions

- These have an important role in describing patterns in searching for strings in many applications (e.g. awk, grep, Perl, ...)

All regular expressions of alphabet \( \Sigma \) are
1. \( \emptyset \) and \( \varepsilon \) are regular expressions,
2. \( a \) is a regular expression of \( \Sigma \) for all \( a \in \Sigma \),
3. if \( R_1 \) and \( R_2 \) are regular expressions, then also
   - \( (R_1 \cup R_2) \),
   - \( (R_1 \cdot R_2) \) and
   - \( R_1^* \)
   are regular expressions

Each regular expression \( R \) of \( \Sigma \) represents a language \( L(R) \)
1. \( L(\emptyset) = \emptyset \),
2. \( L(\varepsilon) = \{\varepsilon\} \),
3. \( L(a) = \{a\} \ \forall \ a \in \Sigma \),
4. \( L((R_1 \cup R_2)) = L(R_1) \cup L(R_2) \),
5. \( L((R_1 \cdot R_2)) = L(R_1) \cdot L(R_2) \) and
6. \( L(R_1^*) = (L(R_1))^* \)

Proper closure: \( R^+ \) is a shorthand for \( RR^* \) (Kleene plus)
Observe: \( R^* \cup \varepsilon = R^* \)
- Let \( R \) be shorthand for the concatenation of \( k \) \( R \)'s with each other.
Examples

- $0^*10^* = \{ w \mid w \text{ contains a single 1 } \}$
- $\Sigma^*001\Sigma^* = \{ w \mid w \text{ contains the string 001 as a substring } \}$
- $1^*(01^*)^* = \{ w \mid \text{ every 0 in } w \text{ is followed by at least one 1 } \}$
- $(\Sigma^\varepsilon)^* = \{ w \mid w \text{ is a string of even length} \}$
- $01 \cup 10 = \{ 01, 10 \}$
- $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{ w \mid w \text{ starts and ends with the same symbol} \}$
- $(0 \cup \varepsilon)^*1^* = 01^*$
- $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{ \varepsilon, 0, 1, 01 \}$
- $1^*\emptyset = \emptyset$
- $\emptyset^* = \{ \varepsilon \}$

For any regular expression $R$

- $R \cup \emptyset = R$ and $R \varepsilon = R$
- However, it may hold that $R \cup \varepsilon \neq R$ and $R \emptyset \neq R$

For example, the unsigned real numbers that can be recognized using the previous automaton can be expressed with the regular expression

$$d^*(d^* \cup \varepsilon)(E(+)\cup(-)\cup\varepsilon)d^* \cup \varepsilon),$$

where $d = (0 \cup \ldots \cup 9)$
Theorem 1.54  A language is regular if and only if some regular expression describes it.

We state and prove both directions of this theorem separately.

Lemma 1.55  If a language is described by a regular expression, then it is regular.

Proof. Any regular expression can be converted into a finite automaton, which recognizes the same language as that described by the regular expression.

There are only six rules by which regular expressions can be composed. The following pictures illustrate the NFA for each of these cases.  □
\[ r = st \]

\[ N_s \rightarrow \varepsilon \rightarrow N_t \]

\[ r = s^* \]

\[ r = s^* \]

\[ N_f \rightarrow \varepsilon \rightarrow N_f \]
Lemma 1.60 If a language is regular, then it is described by a regular expression.

Proof. By definition a regular language can be recognized with a (nondeterministic) finite automaton, which can be converted into a generalized nondeterministic finite automaton (GNFA). The GNFA finally yields a regular expression that is equivalent with the original automaton.

Let $\text{RE}_\Sigma$ denote the set of regular expressions over $\Sigma$. Let $\delta$ denote the transition function of a GNFA, $\delta: Q \times \text{RE}_\Sigma \rightarrow P(Q)$.

$(q, w) \rightarrow (q', w')$ if $q' \in \delta(q, r)$ for some $r \in \text{RE}_\Sigma$ s.t. $w = zw'$, $z \in L(r)$.
A GNFA $M$ can be reduced into a regular expression which describes the language recognized by $M$

1. We compress $M$ into a GNFA with only 2 states (so that the language recognized remains equivalent)
   1. The accept states of $M$ are replaced by a single one (ε arrows)
   2. We remove all other states $q$ except the start state and final state.
      Let $q_i$ and $q_j$ be the predecessor and successor of $q$ on some route passing through $q$.
      Now we can remove $q$ and rename the arrow between $q_i$ and $q_j$ with a new expression.

2. Eventually the GNFA contains at most two states. It is easy to convert the language recognized into a regular expression.
\[(ab \cup (aa \cup b)(ba)^* (bb \cup a))^*\]
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

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

**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
- \( xy^iz \in A \quad \forall i \geq 0, 1, 2, ... \)

**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 $i = 0, 1, 2, ...$

Observe: The pumping lemma does not give us liberty to choose $x$ and $y$ as we please.

Example

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

By the pumping lemma there exists some number $p$ s.t. strings of $L_{parenth}$ 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
- $x = (\ell \quad \ell \leq p - 1$,
- $y = (\ell \quad j \geq 1$, and
- $z = (p^{(p - j)})^p$.

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

because $p - j \neq p$ since $j \geq 1$.

Hence, $L_{parenth}$ 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
  \[ C = \{ \, w \mid \text{w has an equal number of 0s and 1s} \, \} \]
  \[ D = \{ \, w \mid \text{w has an equal number of occurrences of 01 and 10 as substrings} \, \} \]
• 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

An alternative proof for nonregularity of \( C \)

• The complement of a regular language is regular (Exercises 1, question 5)
• The intersection of two regular languages \( A \) and \( B \) can be expressed as
  \[ A \cap B = \overline{A} \cup \overline{B} \]
• Therefore, by Theorem 1.25, the intersection is also regular
• Language \( L_{\text{parenth}} \) can be expressed as the intersection of \( C \) and \( 0^*1^* \), the latter of which is a regular language
• If \( C \) were regular, then its intersection with the regular language \( 0^*1^* \) would also be regular
• However, we know that \( L_{\text{parenth}} \) is not regular
• Therefore, \( C \) cannot either be 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 \( 0 \)'s, so \( xyyz \notin F \).

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

Therefore, \( xy^2z = xyyz = 0^{i+j+i}10^p1 = 0^{i+j}10^p1 \) which does not belong to \( F \) since \( 0^{i+j}1 \) 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 $0^{p+1}1^p$. 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: $xy^0z = xz = 0^{p+1-j}1^p = 0^{p+1-j}1^p \notin E$ 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. $S \rightarrow \epsilon$ and
  2. $S \rightarrow (S)

- These productions generate the strings of the language $L_{parenth}$ starting from the start variable $S$

\[
S^2 \Rightarrow (S)^2 \Rightarrow ((S))^2 \Rightarrow (((S)))^2 \Rightarrow (((((\epsilon)))) = (((())))
\]
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 ⇒ context-free grammar.

One often abbreviates
\[ A \rightarrow w_1 \mid \ldots \mid w_k \]

to describe the alternative productions associated with the variable \( A \).

\[ A \rightarrow w_1, \ldots, A \rightarrow w_k \]

Simple arithmetic expressions
\( (E = \text{expression}, \ T = \text{term and} \ F = \text{factor}) \)

\[
\begin{align*}
E & \rightarrow E + T \mid T \\
T & \rightarrow T \times F \mid F \\
F & \rightarrow (E) \mid a \\
\end{align*}
\]

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

\[
\begin{align*}
E & \Rightarrow T \Rightarrow T \times F \Rightarrow F \times F \Rightarrow (E) \times F \Rightarrow (E + T) \times F \Rightarrow (T + T) \times F \Rightarrow (E + T) \times F \Rightarrow (a + T) \times F \Rightarrow (a + F) \times F \Rightarrow (a + (E)) \times F \Rightarrow (a + (T)) \times F \Rightarrow (a + (F)) \times F \Rightarrow (a + (a)) \times F \Rightarrow (a + (a)) \times a
\end{align*}
\]
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 $uwv$ in grammar $G$, written $uAv \Rightarrow_\epsilon uwv$

- String $u$ derives string $v$ in grammar $G$, written $u \Rightarrow_\epsilon v$, if a sequence $u_1, u_2, \ldots, u_k \in (V \cup \Sigma)^*$ ($k \geq 0$) exists s.t.
- $u \Rightarrow_\epsilon u_1 \Rightarrow_\epsilon u_2 \Rightarrow_\epsilon \ldots \Rightarrow_\epsilon u_k \Rightarrow_\epsilon v$

- $k = 0$: $u \Rightarrow_\epsilon u$ 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

\[
L(G) = \{ w \in \Sigma^* \mid S \Rightarrow_G w \}
\]

• 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 \( A \rightarrow \varepsilon \) 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 \{ $a^i b^j c^k$ $| i = j$ v $j = k$ \} is inherently ambiguous

• An alternative grammar for the simple arithmetic expressions:

$$E \rightarrow E + E \mid E \times E \mid (E) \mid a$$

\[ a + a \times a \]