Automata Theory and Formal Languages

Summer Term 2026

2nd Lecture

Finite Representation of Languages

Prof. Dr. Matthias Mnich *Institute for Algorithms and Complexity*

Finite Representation of Languages

Common formal languages (e.g., C) are infinite, i.e., they contain infinitely many words and sentences.

Finite Representation of Languages

Common formal languages (e.g., C) are infinite, i.e., they contain infinitely many words and sentences.

Consequently, one cannot iterate over (all elements of) those languages in finite time.

Finite Representation of Languages

Common formal languages (e.g., C) are infinite, i.e., they contain infinitely many words and sentences.

Consequently, one cannot iterate over (all elements of) those languages in finite time.

Thus, the *decision problem* "Is $x \in L$?" (for input string $x \in \Sigma$ and fixed L) cannot be solved by brute force.

Finite representation of languages

Goal: Represent a language using finite information.

Finite representation of languages

Goal: Represent a language using finite information.

One way to achieve this goal is the usage of *set notation*

$$S = \{a^n b^m \mid n, m \geq 0\} = \{\varepsilon, a, b, ab, aab, abb, aabb, \dots\}$$

Finite representation of languages

Goal: Represent a language using finite information.

One way to achieve this goal is the usage of *set notation*

$$S = \{a^n b^m \mid n, m \geq 0\} = \{\varepsilon, a, b, ab, aab, abb, aabb, \dots\}$$

But set notation is limited in practice.

Finite representation of languages

Goal: Represent a language using finite information.

One way to achieve this goal is the usage of *set notation*

$$S = \{a^n b^m \mid n, m \geq 0\} = \{\varepsilon, a, b, ab, aab, abb, aabb, \dots\}$$

But set notation is limited in practice.

$\Rightarrow$ In the following we will study a more practical tool.

Regular expressions

A *regular expression* r over an alphabet Σ is defined recursively:

Regular expressions

A *regular expression* r over an alphabet Σ is defined recursively:

- $\emptyset$, ε and each $a \in \Sigma$ are regular expressions, which represent the languages $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ and $L(a) = \{a\}$;

Regular expressions

A *regular expression* r over an alphabet Σ is defined recursively:

- $\emptyset$, ε and each $a \in \Sigma$ are regular expressions, which represent the languages $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ and $L(a) = \{a\}$;
- If r and s are regular expressions then

Regular expressions

A *regular expression* r over an alphabet Σ is defined recursively:

- $\emptyset$, ε and each $a \in \Sigma$ are regular expressions, which represent the languages $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ and $L(a) = \{a\}$;
- If r and s are regular expressions then
  - $(r + s)$ with $L(r + s) = L(r) \cup L(s)$

Regular expressions

A *regular expression* r over an alphabet Σ is defined recursively:

- $\emptyset$, ε and each $a \in \Sigma$ are regular expressions, which represent the languages $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ and $L(a) = \{a\}$;
- If r and s are regular expressions then
  - $(r + s)$ with $L(r + s) = L(r) \cup L(s)$
  - (rs) with $L(rs) = L(r)L(s)$

Regular expressions

A *regular expression* r over an alphabet Σ is defined recursively:

- $\emptyset$, ε and each $a \in \Sigma$ are regular expressions, which represent the languages $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ and $L(a) = \{a\}$;
- If r and s are regular expressions then
  - $(r + s)$ with $L(r + s) = L(r) \cup L(s)$
  - (rs) with $L(rs) = L(r)L(s)$
  - (r^*) with $L(r^*) = L(r)^*$

  are also regular expressions.

Regular expressions

A *regular expression* r over an alphabet Σ is defined recursively:

- $\emptyset$, ε and each $a \in \Sigma$ are regular expressions, which represent the languages $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$ and $L(a) = \{a\}$;
- If r and s are regular expressions then
  - $(r + s)$ with $L(r + s) = L(r) \cup L(s)$
  - (rs) with $L(rs) = L(r)L(s)$
  - (r^*) with $L(r^*) = L(r)^*$

  are also regular expressions.

$L((ab)^*) = \{(ab)^n \mid n \geq 0\}$

Regular languages

The language L over $\Sigma = \{a, b\}$ containing the substring ab is regular, since it can be expressed using the regular expression

$$r = (a + b)^* ab(a + b)^*.$$

Regular languages

The language L over $\Sigma = \{a, b\}$ containing the substring ab is regular, since it can be expressed using the regular expression

$$r = (a + b)^* ab(a + b)^*.$$

A language L that can be described by a regular expression r (i.e. $L(r) = L$) is called *regular*.

Regular languages

The language L over $\Sigma = \{a, b\}$ containing the substring ab is regular, since it can be expressed using the regular expression

$$r = (a + b)^* ab(a + b)^*.$$

A language L that can be described by a regular expression r (i.e. $L(r) = L$) is called *regular*.

There are several important languages that cannot be described by a regular expression (proof later).

Equivalence of regular expressions

Two regular expressions r and s are *equivalent* ($r \equiv s$ or $r \hat{=} s$) if they generate the same language ($L(r) = L(s)$).

Equivalence of regular expressions

Two regular expressions r and s are *equivalent* ($r \equiv s$ or $r \hat{=} s$) if they generate the same language ($L(r) = L(s)$).

For any regular expression r it holds $r^* r \equiv r r^*$, since $L(r^* r) = L(r r^*)$.