Automata Theory and Formal Languages

Summer Term 2026

1st Lecture

Formal Languages

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

Languages

Languages are used for communication between humans and machines (human-human, human-machine, machine-machine).

Languages

Languages are used for communication between humans and machines (human-human, human-machine, machine-machine).

Languages are a medium for transferring information.

Languages

Languages are used for communication between humans and machines (human-human, human-machine, machine-machine).

Languages are a medium for transferring information.

One differentiates between *natural* and *formal* languages:

Languages

Languages are used for communication between humans and machines (human-human, human-machine, machine-machine).

Languages are a medium for transferring information.

One differentiates between *natural* and *formal* languages:

- Natural languages: German, English.

Languages

Languages are used for communication between humans and machines (human-human, human-machine, machine-machine).

Languages are a medium for transferring information.

One differentiates between *natural* and *formal* languages:

- Natural languages: German, English.
- Formal languages: C, Java, HTML, XML.

Natural Languages

- ...work perfectly for human-human communication

Natural Languages

- ...work perfectly for human-human communication (most of the time...).

Natural Languages

- ...work perfectly for human-human communication (most of the time...).
- ...work bad for human-computer and computer-computer communication.

Natural Languages

- ...work perfectly for human-human communication (most of the time...).
- ...work bad for human-computer and computer-computer communication.
- Using heuristics it is possible to understand subsets of natural languages with a computer
  $\Rightarrow$ Machine Learning, Google, Apple, DeepL, ...

Ambiguity in natural languages

Consider the sentence: *We saw her duck.*

Ambiguity in natural languages

Consider the sentence: *We saw her duck.*

The words *her duck* can refer either

Ambiguity in natural languages

Consider the sentence: *We saw her duck.*

Source: Adobe Stock

The words *her duck* can refer either

- to the person's bird (the noun *duck*, modified by the possessive pronoun *her*),

Ambiguity in natural languages

Consider the sentence: *We saw her duck.*

Source: Adobe Stock

The words *her duck* can refer either

- to the person's bird (the noun *duck*, modified by the possessive pronoun *her*),

- or to a motion she made (the verb *duck*, the subject of which is the objective pronoun *her*, object of the verb *saw*)

Source: iStock

Source: Wikipedia

Ambiguity in natural languages

? How to interpret the message?

Ambiguity in natural languages

Source: Wikipedia

How to interpret the message?

- This area is video-monitored by the police to prevent crime.

Ambiguity in natural languages

Source: Wikipedia

How to interpret the message?

- This area is video-monitored by the police to prevent crime.
- This area is video-monitored to prevent crime from the police.

Formal languages

- ... work very well for computer-computer communication.

Formal languages

- ... work very well for computer-computer communication.
- ... work moderately well for human-computer communication.

Formal languages

- ... work very well for computer-computer communication.
- ... work moderately well for human-computer communication.
- ... facilitate theoretical treatment $\Rightarrow$ content of this lecture

Basics of formal languages

An *alphabet* is a non-empty finite set of *characters* (or *letters* or *symbols*).

Basics of formal languages

An *alphabet* is a non-empty finite set of *characters* (or *letters* or *symbols*).

- $\Sigma = \{a, b\}$: alphabet with two characters a and b

Basics of formal languages

An *alphabet* is a non-empty finite set of *characters* (or *letters* or *symbols*).

- $\Sigma = \{a, b\}$: alphabet with two characters a and b
- $\Sigma = \{a, \ldots, z, A, \ldots, Z, 0, \ldots, 9\}$:
  usual alphabet for writing text

Strings

A *word* (or *string*) over the alphabet Σ is a finite sequence $w = a_1 a_2 a_3 \ldots a_n$ of characters from Σ.

Strings

A *word* (or *string*) over the alphabet Σ is a finite sequence $w = a_1 a_2 a_3 \dots a_n$ of characters from Σ.

Convention. We will use small letters to describe strings that are part of a language.

Strings

A *word* (or *string*) over the alphabet Σ is a finite sequence $w = a_1 a_2 a_3 \dots a_n$ of characters from Σ.

Convention. We will use small letters to describe strings that are part of a language.

aa, ab, bba and $baab$ are strings over $\Sigma = \{a, b\}$.

Strings

A *word* (or *string*) over the alphabet Σ is a finite sequence $w = a_1 a_2 a_3 \dots a_n$ of characters from Σ.

Convention. We will use small letters to describe strings that are part of a language.

aa, ab, bba and $baab$ are strings over $\Sigma = \{a, b\}$.

The *length* $|x|$ of a string $x = a_1 a_2 \dots a_n$ is its number $|x| = n$ of characters.

Concatenation of strings

For strings $x = a_1 \dots a_n$ and $y = b_1 \dots b_m$ over alphabets Σ_x and Σ_y, their *concatenation* over the alphabet $\Sigma = \Sigma_x \cup \Sigma_y$ is the string

$$x \circ y = xy = a_1 a_2 \dots a_n b_1 b_2 \dots b_m$$

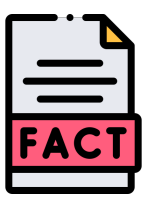
of length $|xy| = n + m$.

Concatenation of strings

For strings $x = a_1 \dots a_n$ and $y = b_1 \dots b_m$ over alphabets Σ_x and Σ_y, their *concatenation* over the alphabet $\Sigma = \Sigma_x \cup \Sigma_y$ is the string

$$x \circ y = xy = a_1 a_2 \dots a_n b_1 b_2 \dots b_m$$

of length $|xy| = n + m$.



concatenation is associative, since
$(x \circ y) \circ z = a_1 a_2 \dots a_n b_1 b_2 \dots b_m c_1 c_2 \dots c_p = x \circ (y \circ z)$.

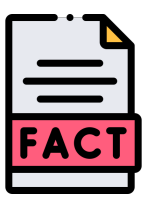
It is thus not necessary to use brackets.

Concatenation of strings

For strings $x = a_1 \dots a_n$ and $y = b_1 \dots b_m$ over alphabets Σ_x and Σ_y, their *concatenation* over the alphabet $\Sigma = \Sigma_x \cup \Sigma_y$ is the string

$$x \circ y = xy = a_1 a_2 \dots a_n b_1 b_2 \dots b_m$$

of length $|xy| = n + m$.



concatenation is associative, since
$(x \circ y) \circ z = a_1 a_2 \dots a_n b_1 b_2 \dots b_m c_1 c_2 \dots c_p = x \circ (y \circ z)$.

It is thus not necessary to use brackets.

concatenation is not commutative, since
$xy = a_1 a_2 \dots a_n b_1 b_2 \dots b_m \neq b_1 b_2 \dots b_m a_1 a_2 \dots a_n = yx$.

The empty string

The *empty string* is the unique string ε of length 0.

The empty string

The *empty string* is the unique string ε of length 0.

The empty string maps any string to itself; it is thus the neutral element of the concatenation operator:

$$\varepsilon x = x = x\varepsilon$$

Exponentiation of strings

The *n^{th} power* x^n of a string x is the $(n-1)$-fold concatenation of x with itself:

$$x^0 := \varepsilon$$

$$x^n := x^{n-1} \circ x \quad \text{for } n \in \mathbb{N} \ .$$

Exponentiation of strings

The *n^{th} power* x^n of a string x is the $(n-1)$-fold concatenation of x with itself:

$$x^0 := \varepsilon$$
$$x^n := x^{n-1} \circ x \quad \text{for } n \in \mathbb{N} \ .$$

- $x^4 = xxxx$
- $(ab)^3 = ababab$

The Kleene star operator

The *Kleene star* (or *Kleene operator* or *Kleene closure*) Σ^* of an alphabet Σ is the set of all strings that can be generated by arbitrary concatenation of its characters; that is,

$$\Sigma^* := \bigcup_{n \geq 0} A_n$$

where A_n is the set of all string combinations of length n.

The Kleene star operator

The *Kleene star* (or *Kleene operator* or *Kleene closure*) Σ^* of an alphabet Σ is the set of all strings that can be generated by arbitrary concatenation of its characters; that is,

$$\Sigma^* := \bigcup_{n \geq 0} A_n$$

where A_n is the set of all string combinations of length n.

For alphabet $\Sigma = \{a, b\}$ it holds
$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$.

The Kleene star operator

The *Kleene star* (or *Kleene operator* or *Kleene closure*) Σ^* of an alphabet Σ is the set of all strings that can be generated by arbitrary concatenation of its characters; that is,

$$\Sigma^* := \bigcup_{n \geq 0} A_n$$

where A_n is the set of all string combinations of length n.

For alphabet $\Sigma = \{a, b\}$ it holds
$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$.

Remarks.

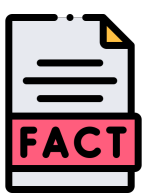
- The same character can be used multiple times.
- The empty string ε is also part of Σ^*.

The Kleene star and Kleene plus operator

The *Kleene star* (or *Kleene operator* or *Kleene closure*) of an alphabet Σ is the set Σ^* of all strings that can be generated by arbitrary concatenation of its characters; that is,

$$\Sigma^* := \bigcup_{n \geq 0} A_n$$

where A_n is the set of all string combinations of length n.



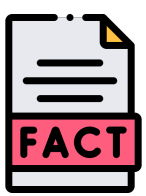
- The set Σ^* is infinite, since we defined Σ to be non-empty.

The Kleene star and Kleene plus operator

The *Kleene star* (or *Kleene operator* or *Kleene closure*) of an alphabet Σ is the set Σ^* of all strings that can be generated by arbitrary concatenation of its characters; that is,

$$\Sigma^* := \bigcup_{n \geq 0} A_n$$

where A_n is the set of all string combinations of length n.



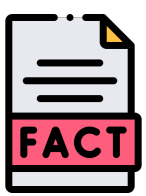
- The set Σ^* is infinite, since we defined Σ to be non-empty.
- It is *countable* and has the same cardinality as the set $\mathbb{N}$ of natural numbers.

The Kleene star and Kleene plus operator

The *Kleene star* (or *Kleene operator* or *Kleene closure*) of an alphabet Σ is the set Σ^* of all strings that can be generated by arbitrary concatenation of its characters; that is,

$$\Sigma^* := \bigcup_{n \geq 0} A_n$$

where A_n is the set of all string combinations of length n.



- The set Σ^* is infinite, since we defined Σ to be non-empty.
- It is *countable* and has the same cardinality as the set $\mathbb{N}$ of natural numbers.

The *Kleene plus* of an alphabet Σ is given by $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$.

Kleene star induces a group structure

Lemma. The structure $(\Sigma^*, \circ)$ is a monoid,
that is, a semigroup with a neutral element.

Kleene star induces a group structure

Lemma. The structure $(\Sigma^*, \circ)$ is a monoid,
that is, a semigroup with a neutral element.

Proof.

- Associativity has already been shown.

Kleene star induces a group structure

Lemma. The structure $(\Sigma^*, \circ)$ is a monoid,
that is, a semigroup with a neutral element.

Proof.

- Associativity has already been shown.
- Existence of a neutral element has already been shown.

Kleene star induces a group structure

Lemma. The structure $(\Sigma^*, \circ)$ is a monoid,
that is, a semigroup with a neutral element.

Proof.

- Associativity has already been shown.
- Existence of a neutral element has already been shown.
- Closure under $\circ$: Let $x \in \Sigma^*$ and $y \in \Sigma^*$ be two strings over the alphabet Σ. Then

$$x \circ y = xy \in \Sigma^* \quad .$$

□

Substrings

- A string x is a *substring* of a string y if

$$y = uxv,$$

  where u and v can be arbitrary strings.
- If $u = \varepsilon$ then x is a *prefix* of y.
- If $v = \varepsilon$ then x is a *suffix* of y.

Substrings

- A string x is a *substring* of a string y if

$$y = uxv,$$

  where u and v can be arbitrary strings.
- If $u = \varepsilon$ then x is a *prefix* of y.
- If $v = \varepsilon$ then x is a *suffix* of y.

For strings x and y. the quantity $|y|_x$ is the number of times that x is a substring of y.

Mirrored strings

For a string $x = a_1 a_2 \ldots a_{n-1} a_n$ of length n, its *mirrored string* is given by

$$x^{\text{Rev}} = a_n a_{n-1} \ldots a_2 a_1 \quad .$$

Formal languages – Definition and examples

A *(formal) language* over the alphabet Σ is a subset L of Σ^*.

Formal languages – Definition and examples

A *(formal) language* over the alphabet Σ is a subset L of Σ^*.

- For alphabet $\Sigma = \{a, b\}$, let L_1 be the set of all strings starting with b, followed by an arbitrary number of a's, and ending with b:

Formal languages – Definition and examples

A *(formal) language* over the alphabet Σ is a subset L of Σ^*.

- For alphabet $\Sigma = \{a, b\}$, let L_1 be the set of all strings starting with b, followed by an arbitrary number of a's, and ending with b:

$$L_1 = \{ba^n b \mid n \in \mathbb{N}_0\}$$

Formal languages – Definition and examples

A *(formal) language* over the alphabet Σ is a subset L of Σ^*.

- For alphabet $\Sigma = \{a, b\}$, let L_1 be the set of all strings starting with b, followed by an arbitrary number of a's, and ending with b:

$$L_1 = \{ba^n b \mid n \in \mathbb{N}_0\}$$

Then $bb \in L_1$, $bab \in L_1$, etc.

Formal languages – Definition and examples

A *(formal) language* over the alphabet Σ is a subset L of Σ^*.

- For alphabet $\Sigma = \{a, b\}$, let L_1 be the set of all strings starting with b, followed by an arbitrary number of a's, and ending with b:

$$L_1 = \{ba^n b \mid n \in \mathbb{N}_0\}$$

Then $bb \in L_1$, $bab \in L_1$, etc.

- For alphabet $\Sigma = \{(,), +, -, *, /, a\}$ let $L_{\text{Bracket}} \subset \Sigma^*$ be the language of all valid bracket expressions.

Formal languages – Definition and examples

A *(formal) language* over the alphabet Σ is a subset L of Σ^*.

- For alphabet $\Sigma = \{a, b\}$, let L_1 be the set of all strings starting with b, followed by an arbitrary number of a's, and ending with b:

$$L_1 = \{ba^n b \mid n \in \mathbb{N}_0\}$$

Then $bb \in L_1$, $bab \in L_1$, etc.

- For alphabet $\Sigma = \{(,), +, -, *, /, a\}$ let $L_{\text{Bracket}} \subset \Sigma^*$ be the language of all valid bracket expressions.

$$(a-a)*a+a/(a+a)-a \in L_{\text{Bracket}}$$

Formal languages – Definition and examples

A *(formal) language* over the alphabet Σ is a subset L of Σ^*.

- For alphabet $\Sigma = \{a, b\}$, let L_1 be the set of all strings starting with b, followed by an arbitrary number of a's, and ending with b:

$$L_1 = \{ba^n b \mid n \in \mathbb{N}_0\}$$

Then $bb \in L_1$, $bab \in L_1$, etc.

- For alphabet $\Sigma = \{(,), +, -, *, /, a\}$ let $L_{\text{Bracket}} \subset \Sigma^*$ be the language of all valid bracket expressions.

$$(a-a)*a+a/(a+a)-a \in L_{\text{Bracket}}$$

$$((aa)) \in L_{\text{Bracket}}$$

Formal languages – Definition and examples

A *(formal) language* over the alphabet Σ is a subset L of Σ^*.

- For alphabet $\Sigma = \{a, b\}$, let L_1 be the set of all strings starting with b, followed by an arbitrary number of a's, and ending with b:

$$L_1 = \{ba^n b \mid n \in \mathbb{N}_0\}$$

Then $bb \in L_1$, $bab \in L_1$, etc.

- For alphabet $\Sigma = \{(,), +, -, *, /, a\}$ let $L_{\text{Bracket}} \subset \Sigma^*$ be the language of all valid bracket expressions.

$$(a-a)*a+a/(a+a)-a \in L_{\text{Bracket}}$$

$$((aa)) \in L_{\text{Bracket}}$$

$$((a+)-a(\notin L_{\text{Bracket}}$$

Concatenation of languages

For languages $X \subset \Sigma_X^*$ over alphabet Σ_X and $Y \subset \Sigma_Y^*$ over alphabet Σ_Y, their *concatenation* is

$$X \circ Y = XY = \{xy \,|\, x \in X \wedge y \in Y\} \ .$$

Concatenation of languages

For languages $X \subset \Sigma_X^*$ over alphabet Σ_X and $Y \subset \Sigma_Y^*$ over alphabet Σ_Y, their *concatenation* is

$$X \circ Y = XY = \{xy \,|\, x \in X \wedge y \in Y\} \ .$$

The concatenation of X and Y thus contains all string combinations where the prefix is a string from X and the suffix is a string from Y.

Concatenation of languages

For languages $X \subset \Sigma_X^*$ over alphabet Σ_X and $Y \subset \Sigma_Y^*$ over alphabet Σ_Y, their *concatenation* is

$$X \circ Y = XY = \{xy \,|\, x \in X \wedge y \in Y\} \ .$$

The concatenation of X and Y thus contains all string combinations where the prefix is a string from X and the suffix is a string from Y.

Convention. Concatenation has higher precedence than set operations ($\cup, \cap$).

Concatenation of languages

For languages $X \subset \Sigma_X^*$ over alphabet Σ_X and $Y \subset \Sigma_Y^*$ over alphabet Σ_Y, their *concatenation* is

$$X \circ Y = XY = \{xy \,|\, x \in X \wedge y \in Y\} \ .$$

The concatenation of X and Y thus contains all string combinations where the prefix is a string from X and the suffix is a string from Y.

Convention. Concatenation has higher precedence than set operations ($\cup, \cap$).

Concatenation of languages

For languages $X \subset \Sigma_X^*$ over alphabet Σ_X and $Y \subset \Sigma_Y^*$ over alphabet Σ_Y, their *concatenation* is

$$X \circ Y = XY = \{xy \,|\, x \in X \wedge y \in Y\} \ .$$

The concatenation of X and Y thus contains all string combinations where the prefix is a string from X and the suffix is a string from Y.

Convention. Concatenation has higher precedence than set operations ($\cup, \cap$).

Exponentiation of languages

The n^{th} *power* of the language $L \subseteq \Sigma^*$ over alphabet Σ is

$$L^n := L^{n-1}L \quad \text{if} \quad n > 0,$$
$$L^0 := \{\varepsilon\} \ .$$

The Kleene star of languages

The *Kleene star* L^* of a language $L \subset \Sigma^*$ is the set of all strings (including the empty string ε) that can be generated by arbitrary concatenation of strings in the language, that is,

$$L^* := \bigcup_{n \geq 0} L^n \ .$$

The Kleene star of languages

The *Kleene star* L^* of a language $L \subset \Sigma^*$ is the set of all strings (including the empty string ε) that can be generated by arbitrary concatenation of strings in the language, that is,

$$L^* := \bigcup_{n \geq 0} L^n \ .$$

For $L = \{01\}$ one has $L^* = \{\varepsilon, 01, 0101, 010101, \dots\}$

The Kleene star of languages

The *Kleene star* L^* of a language $L \subset \Sigma^*$ is the set of all strings (including the empty string ε) that can be generated by arbitrary concatenation of strings in the language, that is,

$$L^* := \bigcup_{n \geq 0} L^n \ .$$

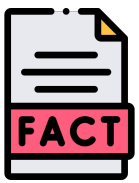
For $L = \{01\}$ one has $L^* = \{\varepsilon, 01, 0101, 010101, \dots\}$
$= \{(01)^n \mid n \geq 0\}$

The Kleene star of languages

The *Kleene star* L^* of a language $L \subset \Sigma^*$ is the set of all strings (including the empty string ε) that can be generated by arbitrary concatenation of strings in the language, that is,

$$L^* := \bigcup_{n \geq 0} L^n \ .$$

For $L = \{01\}$ one has $L^* = \{\varepsilon, 01, 0101, 010101, \dots\}$
$= \{(01)^n \mid n \geq 0\}$



Observations.

- The set L^* usually is infinite with just two exceptions.
- It is *countable* and has the same cardinality as the set $\mathbb{N}$ of natural numbers.