

# Introduction to Theoretical Computer Science

Zdeněk Sawa

Department of Computer Science, FEI,  
Technical University of Ostrava  
17. listopadu 2172/15, Ostrava-Poruba 708 00  
Czech republic

February 19, 2024

Name: doc. Ing. Zdeněk Sawa, Ph.D.

E-mail: `zdenek.sawa@vsb.cz`

Room: EA413

Web: <https://www.cs.vsb.cz/sawa/uti/index-en.html>

On these pages you will find:

- Information about the course
- Study texts
- Slides from lectures
- Exercises for tutorials
- Recent news for the course
- A link to a page with animations

- **Credit** (30 points):
  - Written test (24 points) — it will be written on a tutorial
    - The minimal requirement for obtaining the credit is 12 points.
    - A correcting test for 20 points.
  - Activity on tutorials (6 points)
    - The minimal requirement for obtaining the credit is 3 points.
- **Exam** (70 points)
  - A written exam consisting of two parts (35 points for each part); it is necessary to obtain at least 12 points for each part.
  - It is necessary to obtain at least 30 points.

**Theoretical computer science** — a scientific field on the border between computer science and mathematics

- investigation of general questions concerning algorithms and computations
- study of different kinds of formalisms for description of algorithms
- study of different approaches for description of syntax and semantics of formal languages (mainly programming languages)
- a mathematical approach to analysis and solution of problems (proofs of general mathematical propositions concerning algorithms)

Examples of some typical questions studied in theoretical computer science:

- Is it possible to solve the given problem using some algorithm?
- If the given problem can be solved by an algorithm, what is the computational complexity of this algorithm?
- Is there an efficient algorithm solving the given problem?
- How to check that a given algorithm is really a correct solution of the given problem?
- What kinds instructions are sufficient for a given machine to perform a given algorithm?

# Algorithms and Problems

**Algorithm** — mechanical procedure that computes something (it can be executed by a computer)

Algorithms are used for solving **problems**.

An example of an algorithmic problem:

**Input:** Natural numbers  $x$  and  $y$ .

**Output:** Natural number  $z$  such that  $z = x + y$ .

# Algorithms and Problems

**Algorithm** — mechanical procedure that computes something (it can be executed by a computer)

Algorithms are used for solving **problems**.

An example of an algorithmic problem:

**Input:** Natural numbers  $x$  and  $y$ .

**Output:** Natural number  $z$  such that  $z = x + y$ .

A particular input of a problem is called an **instance** of the problem.

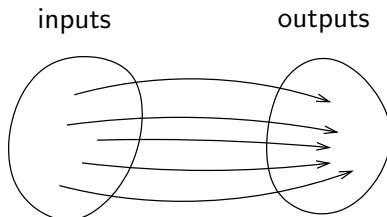
**Example:** An example of an instance of the problem given above is a pair of numbers 728 and 34.

The corresponding output for this instance is number 762.

## Problem

When specifying a **problem** we must determine:

- what is the set of possible inputs
- what is the set of possible outputs
- what is the relationship between inputs and outputs





## Problem “Sorting”

**Input:** A sequence of elements  $a_1, a_2, \dots, a_n$ .

**Output:** Elements of the sequence  $a_1, a_2, \dots, a_n$  ordered from the least to the greatest.

### Example:

- Input: 8, 13, 3, 10, 1, 4
- Output: 1, 3, 4, 8, 10, 13

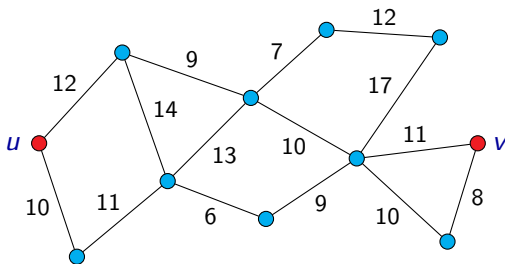
# An example of an algorithmic problem

## Problem “Finding the shortest path in an (undirected) graph”

**Input:** An undirected graph  $G = (V, E)$  with edges labelled with numbers, and a pair of nodes  $u, v \in V$ .

**Output:** The shortest path from node  $u$  to node  $v$ .  
(Or information that there is no such path.)

### Example:



# Algorithms and Problems

An algorithm **solves** a given problem if:

- For each input, the computation of the algorithm halts after a finite number of steps.
- For each input, the algorithm produces a correct output.

**Correctness** of an algorithm — verifying that the algorithm really solves the given problem

**Computational complexity** of an algorithm:

- **time complexity** — how the running time of the algorithm depends on the size of input data
- **space complexity** — how the amount of memory used by the algorithm depends on the size of input data

**Remark:** For one problem there can be many different algorithms that correctly solve the problem.

## Problem “Primality”

**Input:** A natural number  $n$ .

**Output:** YES if  $n$  is a prime, NO otherwise.

**Remark:** A natural number  $n$  is a **prime** if it is greater than 1 and is divisible only by numbers 1 and  $n$ .

Few of the first primes: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, ...

The problems, where the set of outputs is  $\{\text{YES}, \text{NO}\}$  are called **decision problems**.

Decision problems are usually specified in such a way that instead of describing what the output is, a question is formulated.

## Example:

### Problem “Primality”

**Input:** A natural number  $n$ .

**Question:** Is  $n$  a prime?

# Optimization Problems

Those problems where for each input instance there is a corresponding set of **feasible solutions** and where the aim is to select between these feasible solutions that is some respect minimal or maximal (or possibly to find out that there are no feasible solutions), are called **optimization problems**.

## Example:

Problem “Finding the shortest path in an (undirected) graph”

**Input:** An undirected graph  $G = (V, E)$  with edges labelled with numbers, and a pair of nodes  $u, v \in V$ .

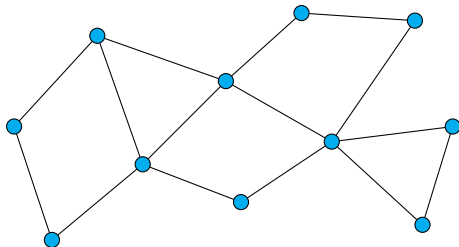
**Output:** The shortest path from node  $u$  to node  $v$ .

# Optimization Problems

## Problem “Coloring of a graph”

**Input:** An undirected graph  $G$ .

**Output:** The minimal number of colors to color the nodes of the graph  $G$  in such a way that no two nodes connected with an edge are colored with the same color, and a concrete example of such coloring using this minimal number of colors.

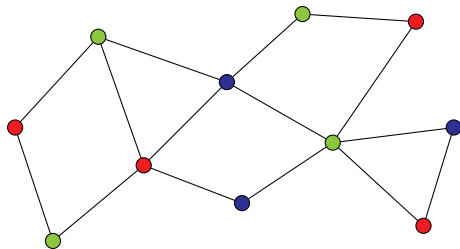


# Optimization Problems

## Problem “Coloring of a graph”

**Input:** An undirected graph  $G$ .

**Output:** The minimal number of colors to color the nodes of the graph  $G$  in such a way that no two nodes connected with an edge are colored with the same color, and a concrete example of such coloring using this minimal number of colors.



Colors: 3

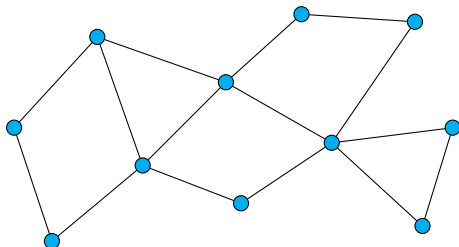


# Optimization Problems

## Problem “Coloring of a graph with $k$ colors”

**Input:** An undirected graph  $G$  and a natural number  $k$ .

**Question:** Is it possible to color the nodes of the graph  $G$  with  $k$  colors in such a way that no two nodes connected with an edge are colored with the same color?



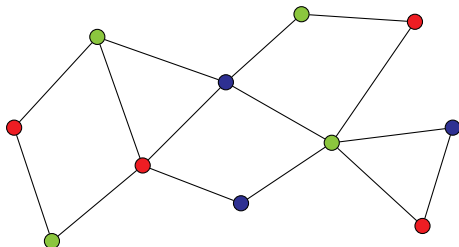
$k = 3$

# Optimization Problems

## Problem “Coloring of a graph with $k$ colors”

**Input:** An undirected graph  $G$  and a natural number  $k$ .

**Question:** Is it possible to color the nodes of the graph  $G$  with  $k$  colors in such a way that no two nodes connected with an edge are colored with the same color?



$k = 3$

# Algorithmically Solvable Problems

Let us assume we have a problem  $P$ .

If there is an algorithm solving the problem  $P$  then we say that the problem  $P$  is **algorithmically solvable**.

If  $P$  is a decision problem and there is an algorithm solving the problem  $P$  then we say that the problem  $P$  is **decidable (by an algorithm)**.

If we want to show that a problem  $P$  is algorithmically solvable, it is sufficient to show some algorithm solving it (and possibly show that the algorithm really solves the problem  $P$ ).

# Algorithmically Unsolvable Problems

A problem that is not algorithmically solvable is **algorithmically unsolvable**.

A decision problem that is not decidable is **undecidable**.

Surprisingly, there are many (exactly defined) problems, for which it was proved that they are not algorithmically solvable.

**Computability theory** — area of theoretical computer science studying, which problems can be solved algorithmically and which cannot.

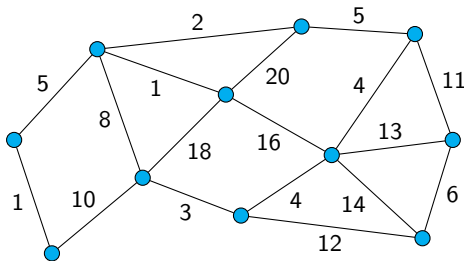
# Complexity Theory

Many problems are algorithmically solvable but there do not exist (or are not known) efficient algorithms solving them:

## TSP - traveling salesman problem

**Input:** An undirected graph  $G$  with edges labelled with natural numbers.

**Output:** A shortest closed path that goes through all vertices of the graph.



Some other areas of theoretical computer science:

- complexity theory
- theory of formal languages
- models of computation
- parallel and distributed algorithms
- ...

# Theory of Formal Languages

An area of theoretical computer science dealing with questions concerning **syntax**.

- **Language** — a set of words
- **Word** — a sequences of symbols from some alphabet
- **Alphabet** — a set of **symbols** (or **letters**)

Words and languages appear in computer science on many levels:

- Representation of input and output data
- Representation of programs
- Manipulation with character strings or files
- ...

Examples of problem types, where theory of formal languages is useful:

- Construction of compilers:
  - Lexical analysis
  - Syntactic analysis
  
- Searching in text:
  - Searching for a given text pattern
  - Searching for a part of text specified by a regular expression



- **Alphabet** — a nonempty finite set of **symbols**

**Example:**  $\Sigma = \{a, b, c, d\}$

- **Word** — a finite sequence of symbols from the given alphabet

**Example:** `cabcbbba`

The set of all words of alphabet  $\Sigma$  is denoted with  $\Sigma^*$ .

For variables, whose values are words, we will use names such as  $w, u, v, x, y, z$ , etc., possibly with indexes (e.g.,  $w_1, w_2$ )

So when we write  $w = \text{cabcbbba}$ , it means that the value of variable  $w$  is word `cabcbbba`.

Similarly, the notation  $w \in \Sigma^*$  means that the value of a variable  $w$  is some word consisting of symbols belonging to alphabet  $\Sigma$ .

## Definition

A **(formal) language**  $L$  over an alphabet  $\Sigma$  is a subset of  $\Sigma^*$ , i.e.,  $L \subseteq \Sigma^*$ .

**Example:** Let us assume that  $\Sigma = \{a, b, c\}$ :

- Language  $L_1 = \{aab, bcca, aaaaa\}$
- Language  $L_2 = \{w \in \Sigma^* \mid \text{the number of occurrences of } b \text{ in } w \text{ is even}\}$

## Example:

Alphabet  $\Sigma$  is the set of all ASCII characters.

Example of a word:

```
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

#include\_<stdio.h> ↔↔ int\_<main() ↔ { ↔ \_<<<<<printf("He...

Formalisms used for description of formal languages:

- automata
- grammars
- regular expressions

# Encoding of Input and Output

Inputs and outputs of an algorithm could be encoded as words over some alphabet  $\Sigma$ .

**Example:** For example, for problem “Sorting” we can take alphabet  $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, , \}$ .

An example of input data (as a word over alphabet  $\Sigma$ ):

826,13,3901,128,562

and the corresponding output data (as a word over alphabet  $\Sigma$ )

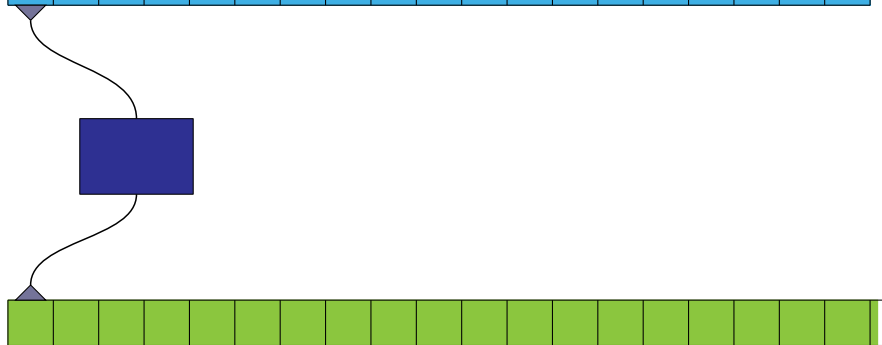
13,128,562,826,3901

**Remark:** It is often the case that only some words over the given alphabet represent valid input or output.

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input

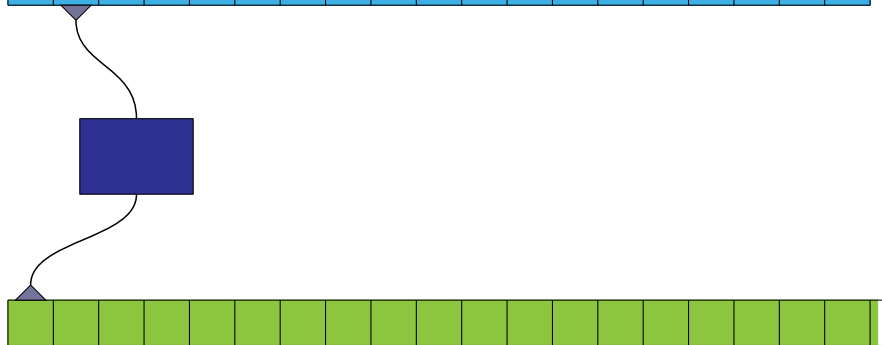


Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output



# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input

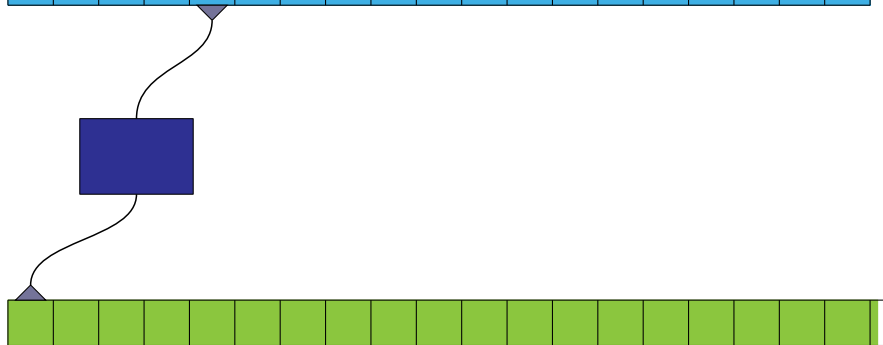


Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output



# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input

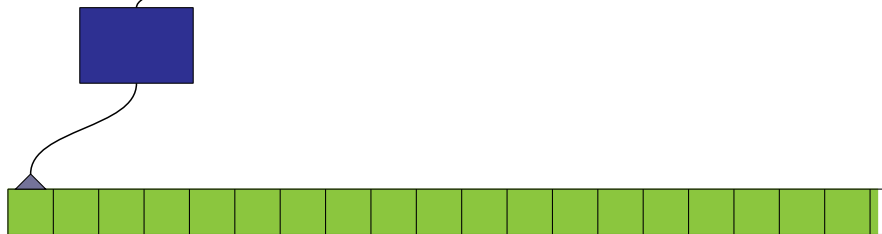


Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input

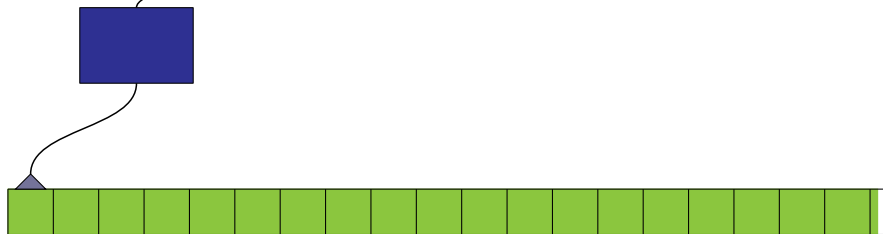


Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output



# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input

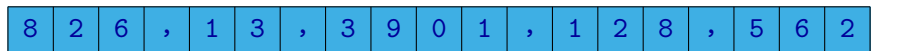


Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output



# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input

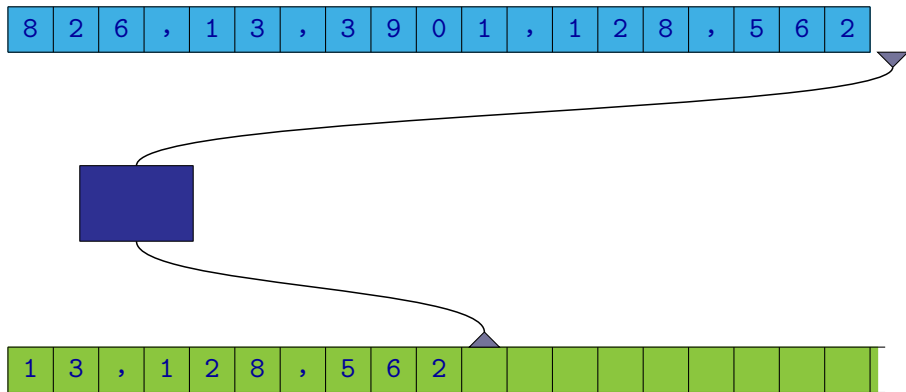


Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input

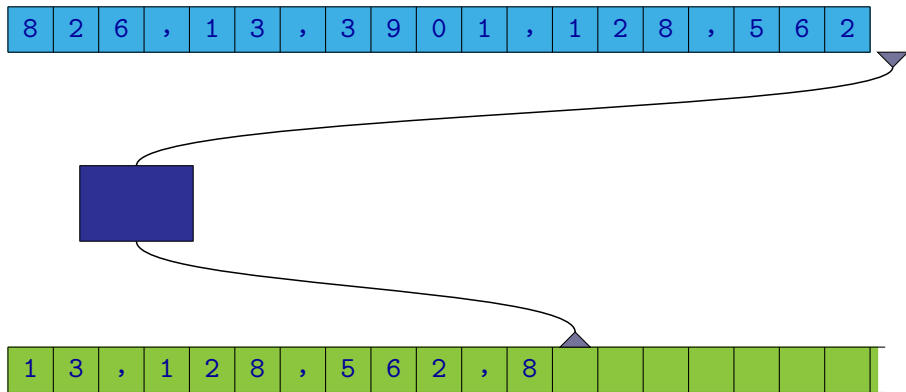


Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input

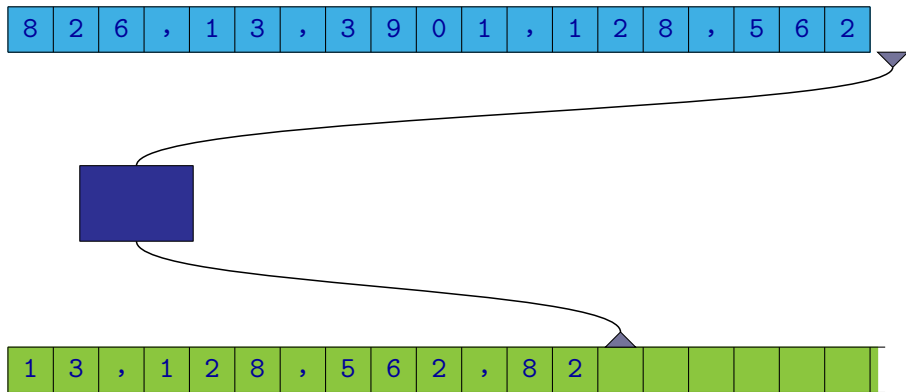


Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input

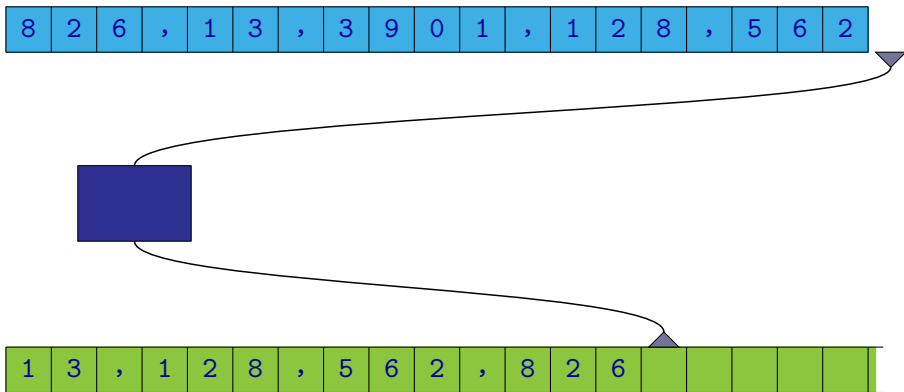


Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output



# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input



Output

# Input/Output Behaviour of an Algorithm

We can assume that the algorithm is executed on a certain type of machine.

Input

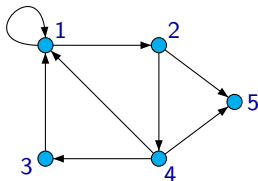


Output

# Encoding of Input and Output

**Example:** If an input for a given problem is graph, it could be represented as a pair of two lists — a list of nodes and a list of edges:

For example, the following graph



could be represented as a word

$(1, 2, 3, 4, 5), ((1, 2), (2, 4), (4, 3), (3, 1), (1, 1), (2, 5), (4, 5), (4, 1))$

over alphabet  $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ,, (, )\}$ .

# Algorithms for Decision Problems

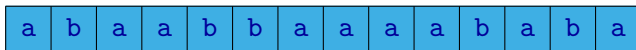
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input



# Algorithms for Decision Problems

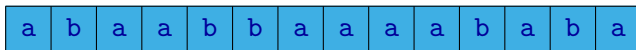
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input



# Algorithms for Decision Problems

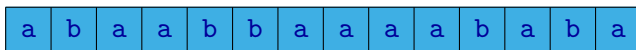
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input





# Algorithms for Decision Problems

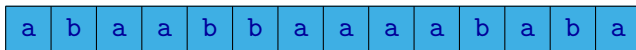
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input



# Algorithms for Decision Problems

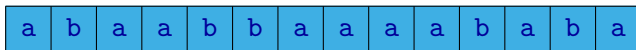
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input



# Algorithms for Decision Problems

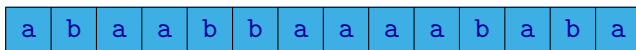
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input



# Algorithms for Decision Problems

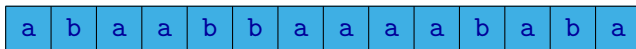
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input



# Algorithms for Decision Problems

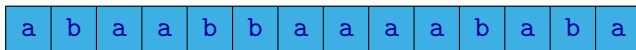
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input



# Algorithms for Decision Problems

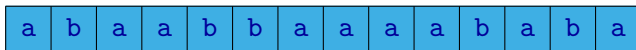
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input



# Algorithms for Decision Problems

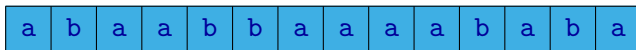
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input



# Algorithms for Decision Problems

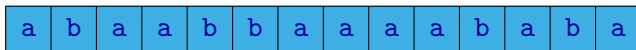
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input





# Algorithms for Decision Problems

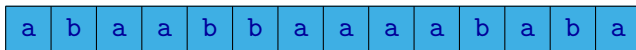
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input



# Algorithms for Decision Problems

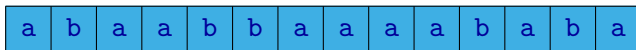
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input



# Algorithms for Decision Problems

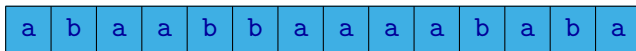
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input



# Algorithms for Decision Problems

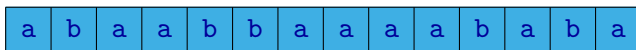
In the case of an algorithm that solves some **decision** problem it is sufficient that the algorithm just provides an answer **YES** or **NO**.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Input



No

# Correspondence between Recognizing Formal Languages and Decision Problems

There is a close correspondence between recognizing words from a given language and decision problems:

- For each language  $L$  over some alphabet  $\Sigma$  there is a corresponding decision problem:

**Input:** A word  $w$  over alphabet  $\Sigma$ .

**Question:** Does  $w$  belong to  $L$ ?

- For each decision problem  $P$  where inputs are encoded as words over alphabet  $\Sigma$  there is a corresponding language:

The language  $L$  containing of exactly those words  $w$  over alphabet  $\Sigma$ , for which the answer to the question stated in problem  $P$  is “YES”.

# Correspondence between Recognizing Formal Languages and Decision Problems

**Example:** The following decision problem can be viewed as the language  $L$  given below and vice versa.

## Problem

**Input:** A word  $w$  over alphabet  $\{a, b\}$ .

**Question:** Does the word  $w$  contain an even number of occurrences of symbol  $b$ ?

Language

$L = \{ w \in \{a, b\}^* \mid w \text{ contains an even number of occurrences of symbol } b \}$

# Models of Computation

We can consider different types of machines that are able to perform an algorithm.

There can be many different kinds of differences between these types of machines:

- what types of instructions they can execute
- what types of data they can store in their memory and this memory is organised
- ...

Different kinds of such machines are called **models of computation**.

In the case of very simple kinds of such machines they are usually called **automata** in the formal language theory.

In this course we will see several types of such automata.

For different types of models of computation analyse for example:

- what algorithmic problems can be solved by such machines and what languages they can recognise.
- how efficiently they can execute different algorithms
- how machines of a certain type can simulate the computations of some other type of machines
- how the number of instructions that are executed by the machine in such simulation grows compared to the original machine
- ...



# Formal Languages

# Alphabet and Word

## Definition

**Alphabet** is a nonempty finite set of **symbols**.

**Remark:** An alphabet is often denoted by the symbol  $\Sigma$  (upper case sigma) of the Greek alphabet.

## Definition

A **word** over a given alphabet is a finite sequence of symbols from this alphabet.

### Example 1:

$\Sigma = \{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z\}$

Words over alphabet  $\Sigma$ :      HELLO      XYZZY      COMPUTER

## Example 2:

$\Sigma_2 = \{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, \_ \}$

A word over alphabet  $\Sigma_2$ : HELLO\_WORLD

## Example 3:

$\Sigma_3 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Words over alphabet  $\Sigma_3$ : 0, 31415926536, 65536

## Example 4:

Words over alphabet  $\Sigma_4 = \{0, 1\}$ : 011010001, 111, 1010101010101010

## Example 5:

Words over alphabet  $\Sigma_5 = \{a, b\}$ : aababb, abbabbba, aaab

The **set of all words** over alphabet  $\Sigma$  is denoted  $\Sigma^*$ .

## Definition

A **(formal) language**  $L$  over an alphabet  $\Sigma$  is a subset of  $\Sigma^*$ , i.e.,  $L \subseteq \Sigma^*$ .

**Example 1:** The set  $\{00, 01001, 1101\}$  is a language over alphabet  $\{0, 1\}$ .

**Example 2:** The set of all syntactically correct programs in the C programming language is a language over the alphabet consisting of all ASCII characters.

**Example 3:** The set of all texts containing the sequence `hello` is a language over alphabet consisting of all ASCII characters.

# Some Basic Concepts

The **length of a word** is the number of symbols of the word.

For example, the length of word `abaab` is 5.

The length of a word  $w$  is denoted  $|w|$ .

For example, if  $w = \text{abaab}$  then  $|w| = 5$ .

We denote the number of occurrences of a symbol  $a$  in a word  $w$  by  $|w|_a$ .

**Example:** If  $w = \text{cabcbba}$  then  $|w| = 7$ ,  $|w|_a = 2$ ,  $|w|_b = 3$ ,  $|w|_c = 2$ ,  $|w|_d = 0$ .

An **empty word** is a word of length 0, i.e., the word containing no symbols.

The empty word is denoted by the letter  $\varepsilon$  (epsilon) of the Greek alphabet.

$$|\varepsilon| = 0$$

# Concatenation of Words

One of operations we can do on words is the operation of **concatenation**: For example, the concatenation of words `cabc` and `bba` is the word `cabcba`.

The operation of concatenation is denoted by symbol  $\cdot$  (it is similar to multiplication). This symbol can be omitted.

So, for  $u, v \in \Sigma^*$ , the concatenation of words  $u$  and  $v$  is written as  $u \cdot v$  or just  $uv$ .

**Example:** If  $u = cabc$  and  $v = bba$ , then

$$u \cdot v = cabcba$$

**Remark:** Formally, the concatenation of words over alphabet  $\Sigma$  is a function of type

$$\Sigma^* \times \Sigma^* \rightarrow \Sigma^*$$

# Concatenation of Words

Concatenation is **associative**, i.e., for every three words  $u$ ,  $v$ , and  $w$  we have

$$(u \cdot v) \cdot w = u \cdot (v \cdot w)$$

which means that we can omit parenthesis when we write multiple concatenations. For example, we can write  $w_1 \cdot w_2 \cdot w_3 \cdot w_4 \cdot w_5$  instead of  $(w_1 \cdot (w_2 \cdot w_3)) \cdot (w_4 \cdot w_5)$ .

Word  $\varepsilon$  is a neutral element for the operation of concatenation, so for every word  $w$  we also have:

$$\varepsilon \cdot w = w \cdot \varepsilon = w$$

**Remark:** It is obvious that if the given alphabet contains at least two different symbols, the operation of concatenation is not commutative, e.g.,

$$a \cdot b \neq b \cdot a$$

# Power of a Word

For arbitrary word  $w \in \Sigma^*$  and arbitrary  $k \in \mathbb{N}$  we can define word  $w^k$  as the word obtained by concatenating  $k$  copies of the word  $w$ .

**Example:** For  $w = abb$  it is  $w^4 = abbabbabbabb$ .

**Example:** Notation  $a^5 b^3 a^4$  denotes word  $aaaaabbbaaaa$ .

A little bit more formal definition looks as follows:

$$w^0 = \varepsilon, \quad w^{k+1} = w^k \cdot w \quad \text{for } k \in \mathbb{N}$$

This means

$$\begin{aligned} w^0 &= \varepsilon \\ w^1 &= w \\ w^2 &= w \cdot w \\ w^3 &= w \cdot w \cdot w \\ w^4 &= w \cdot w \cdot w \cdot w \\ w^5 &= w \cdot w \cdot w \cdot w \cdot w \\ &\dots \end{aligned}$$



# Reverse of a Word

The **reverse** of a word  $w$  is the word  $w$  written from backwards (in the opposite order).

The reverse of a word  $w$  is denoted  $w^R$ .

**Example:**  $w = \text{abbab}$        $w^R = \text{babba}$

So if  $w = a_1a_2\cdots a_n$  (where  $a_i \in \Sigma$ ) then  $w^R = a_na_{n-1}\cdots a_1$ .

We can define  $w^R$  using the following inductively defined function  $\text{rev} : \Sigma^* \rightarrow \Sigma^*$  as the value  $\text{rev}(w)$ .

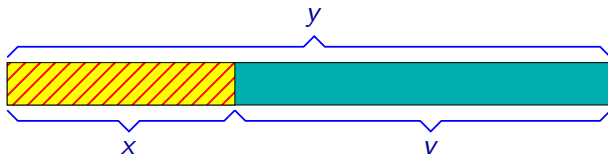
The function  $\text{rev}$  is defined as follows:

- $\text{rev}(\varepsilon) = \varepsilon$
- for  $a \in \Sigma$  and  $w \in \Sigma^*$  it holds that  $\text{rev}(a \cdot w) = \text{rev}(w) \cdot a$

# Prefix of a Word

## Definition

A word  $x$  is a **prefix** of a word  $y$  if there exists a word  $v$  such that  $y = xv$ .

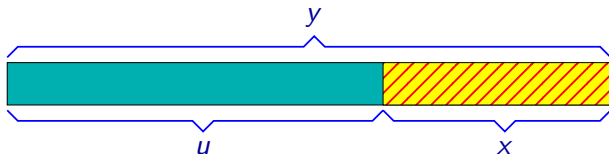


**Example:** Prefixes of the word  $abaab$  are  $\epsilon$ ,  $a$ ,  $ab$ ,  $aba$ ,  $abaa$ ,  $abaab$ .

# Suffix of a Word

## Definition

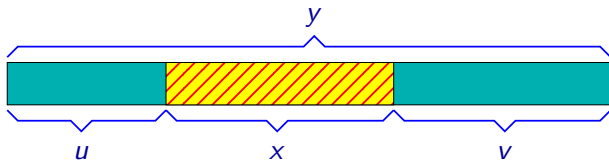
A word  $x$  is a **suffix** of a word  $y$  if there exists a word  $u$  such that  $y = ux$ .



**Example:** Suffixes of the word `abaab` are  $\epsilon$ , `b`, `ab`, `aab`, `baab`, `abaab`.

## Definition

A word  $x$  is a **subword** of a word  $y$  if there exist words  $u$  and  $v$  such that  $y = uxv$ .

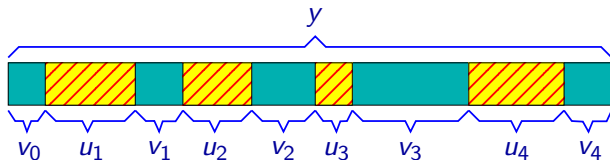


**Example:** Subwords of the word  $abaab$  are  $\varepsilon$ ,  $a$ ,  $b$ ,  $ab$ ,  $ba$ ,  $aa$ ,  $aba$ ,  $baa$ ,  $aab$ ,  $abaa$ ,  $baab$ ,  $abaab$ .

# Subsequence

## Definition

A word  $x$  is a **subsequence** of a word  $y$  if there is a number  $n$  and words  $u_1, u_2, \dots, u_n$  and  $v_0, v_1, \dots, v_n$  such that  $x = u_1 u_2 \dots u_n$  and  $y = v_0 u_1 v_1 u_2 v_2 \dots u_n v_n$ .



**Example:** Word  $cbab$  is a subsequence of word  $acabccabbaa$ .

# Order on Words

Let us assume some (linear) order  $<$  on the symbols of alphabet  $\Sigma$ , i.e., if  $\Sigma = \{a_1, a_2, \dots, a_n\}$  then

$$a_1 < a_2 < \dots < a_n.$$

**Example:**  $\Sigma = \{a, b, c\}$  with  $a < b < c$ .

The following (linear) order  $<_L$  can be defined on  $\Sigma^*$ :

$x <_L y$  iff:

- $|x| < |y|$ , or
- $|x| = |y|$  there exist words  $u, v, w \in \Sigma^*$  and symbols  $a, b \in \Sigma$  such that

$$x = uav \quad y = ubw \quad a < b$$

Informally, we can say that in order  $<_L$  we order words according to their length, and in case of the same length we order them lexicographically.

# Order on Words

All words over alphabet  $\Sigma$  can be ordered by  $<_L$  into a sequence

$$w_0, w_1, w_2, \dots$$

where every word  $w \in \Sigma^*$  occurs exactly once, and where for each  $i, j \in \mathbb{N}$  it holds that  $w_i <_L w_j$  iff  $i < j$ .

**Example:** For alphabet  $\Sigma = \{a, b, c\}$  (where  $a < b < c$ ), the initial part of the sequence looks as follows:

$\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, aac, aba, abb, abc, \dots$

For example, when we talk about the first ten words of a language  $L \subseteq \Sigma^*$ , we mean ten words that belong to language  $L$  and that are smallest of all words of  $L$  according to order  $<_L$ .

$\epsilon$   
a  
b  
c  
aa  
ab  
ac  
ba  
bb  
bc  
ca  
cb  
cc  
aaa  
aab  
aac  
aba  
abb  
abc  
⋮

## Example:

Language

$$L = \{ w \in \{a, b, c\}^* \mid |w|_b \bmod 2 = 0 \}$$



# Order on Words

ε	1
a	2
b	
c	3
aa	4
ab	
ac	5
ba	
bb	6
bc	
ca	7
cb	
cc	8
aaa	9
aab	
aac	10
aba	
abb	11
abc	
⋮	

**Example:**

Language

$$L = \{ w \in \{a, b, c\}^* \mid |w|_b \bmod 2 = 0 \}$$

# Operations on Languages

Let us say we have already described some languages. We can create new languages from these languages using different **operations on languages**.

So a description of a complicated language can be decomposed in such a way that it is described a result of an application of some operations on some simpler languages.

Examples of important operations on languages:

- union
- intersection
- complement
- concatenation
- iteration
- ...

**Remark:** It is assumed the languages involved in these operations use the same alphabet  $\Sigma$ .

# Set Operations on Languages

Since languages are sets, we can apply any set operations to them:

**Union** –  $L_1 \cup L_2$  is the language consisting of the words belonging to language  $L_1$  or to language  $L_2$  (or to both of them).

**Intersection** –  $L_1 \cap L_2$  is the language consisting of the words belonging to language  $L_1$  and also to language  $L_2$ .

**Complement** –  $\overline{L_1}$  is the language containing those words from  $\Sigma^*$  that do not belong to  $L_1$ .

**Difference** –  $L_1 - L_2$  is the language containing those words of  $L_1$  that do not belong to  $L_2$ .

**Remark:** We assume that  $L_1, L_2 \subseteq \Sigma^*$  for some given alphabet  $\Sigma$ .

# Set Operations on Languages

Formally:

**Union:**  $L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$

**Intersection:**  $L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \in L_2\}$

**Complement:**  $\overline{L_1} = \{w \in \Sigma^* \mid w \notin L_1\}$

**Difference:**  $L_1 - L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \notin L_2\}$

# Set Operations on Languages

## Example:

Consider languages over alphabet  $\{a, b\}$ .

- $L_1$  — the set of all words containing subword **baa**
- $L_2$  — the set of all words with an even number of occurrences of symbol **b**

Then

- $L_1 \cup L_2$  — the set of all words containing subword **baa** or an even number of occurrences of **b**
- $L_1 \cap L_2$  — the set of all words containing subword **baa** and an even number of occurrences of **b**
- $\overline{L_1}$  — the set of all words that do not contain subword **baa**
- $L_1 - L_2$  — the set of all words that contain subword **baa** but do not contain an even number of occurrences of **b**

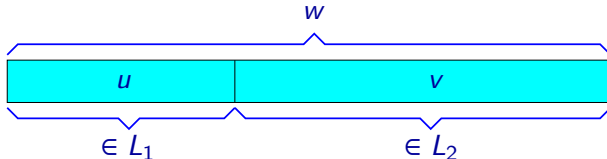
# Concatenation of Languages

## Definition

**Concatenation of languages**  $L_1$  and  $L_2$ , where  $L_1, L_2 \subseteq \Sigma^*$ , is the language  $L \subseteq \Sigma^*$  such that for each  $w \in \Sigma^*$  it holds that

$$w \in L \iff (\exists u \in L_1)(\exists v \in L_2)(w = u \cdot v)$$

The concatenation of languages  $L_1$  and  $L_2$  is denoted  $L_1 \cdot L_2$ .



# Concatenation of Languages

## Example:

$$\begin{aligned}L_1 &= \{abb, ba\} \\L_2 &= \{a, ab, bbb\}\end{aligned}$$

The language  $L_1 \cdot L_2$  contains the following words:

abba    abbab    abbbbb    baa    baab    babbb

**Remark:** Note that the concatenation of languages is associative, i.e., for arbitrary languages  $L_1, L_2, L_3$  it holds that:

$$L_1 \cdot (L_2 \cdot L_3) = (L_1 \cdot L_2) \cdot L_3$$

# Power of a Language

Notation  $L^k$ , where  $L \subseteq \Sigma^*$  and  $k \in \mathbb{N}$ , denotes the concatenation of the form

$$L \cdot L \cdot \dots \cdot L$$

where the language  $L$  occurs  $k$  times, i.e.,

$$\begin{aligned} L^0 &= \{\varepsilon\} \\ L^1 &= L \\ L^2 &= L \cdot L \\ L^3 &= L \cdot L \cdot L \\ L^4 &= L \cdot L \cdot L \cdot L \\ L^5 &= L \cdot L \cdot L \cdot L \cdot L \\ &\dots \end{aligned}$$

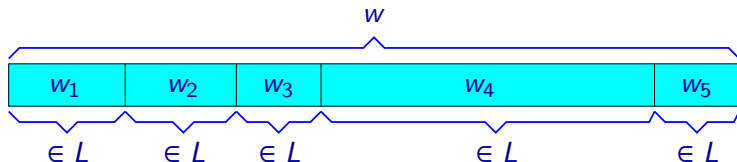
**Example:** For  $L = \{aa, b\}$ , the language  $L^3$  contains the following words:

aaaaaa   aaaab   aabaa   aabb   baaaa   baab   bbaa   bbb



# Power of a Language

**Example:** A word in language  $L^5$  is created by concatenating five words from language  $L$ :



Formally, the  $k$ -th power of a language  $L$ , denoted  $L^k$  can be defined using the following inductive definition:

$$L^0 = \{\varepsilon\}, \quad L^{k+1} = L^k \cdot L \quad \text{for } k \in \mathbb{N}$$

# Iteration of a Language

The **iteration of a language**  $L$ , denoted  $L^*$ , is the language consisting of words created by concatenation of some arbitrary number of words from language  $L$ .

I.e., a word  $w$  belongs to  $L^*$  iff there exists a sequence  $w_1, w_2, \dots, w_n$  of words from language  $L$  such that

$$w = w_1 w_2 \cdots w_n .$$

**Example:**  $L = \{aa, b\}$

$$L^* = \{\varepsilon, aa, b, aaaa, aab, baa, bb, aaaaaa, aaaab, aabaa, aabb, \dots\}$$

**Remark:** The number of concatenated words can be 0, which means that  $\varepsilon \in L^*$  always holds (it does not matter if  $\varepsilon \in L$  or not).

# Iteration of a Language

Formally, the language  $L^*$  can be defined as the union of all powers of language  $L$ . I.e., a word  $w$  belongs to the language  $L^*$  iff if there exists  $k \in \mathbb{N}$  such that  $w \in L^k$ :

## Definition

The **iteration of a language**  $L$  is the language

$$L^* = \bigcup_{k \geq 0} L^k$$

**Remark:**

$$\bigcup_{k \geq 0} L^k = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \dots$$

# Iteration of a Language

Notation  $L^+$  denotes the language consisting of those words that can be created as a concatenation of a non-zero number of words from language  $L$ .

So it holds that

$$L^+ = \bigcup_{k \geq 1} L^k$$

i.e.

$$L^+ = L^1 \cup L^2 \cup L^3 \cup \dots$$

Formally, the language  $L^+$  can be defined also as follows:

$$L^+ = L \cdot L^*$$

The **reverse** of a language  $L$  is the language consisting of reverses of all words of  $L$ .

Reverse of a language  $L$  is denoted  $L^R$ .

$$L^R = \{w^R \mid w \in L\}$$

**Example:**  $L = \{ab, baaba, aaab\}$   
 $L^R = \{ba, abaab, baaa\}$

# Some Properties of Operations on Languages

$$L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$$

$$L_1 \cup L_2 = L_2 \cup L_1$$

$$L_1 \cup L_1 = L_1$$

$$L_1 \cup \emptyset = L_1$$

$$L_1 \cap (L_2 \cap L_3) = (L_1 \cap L_2) \cap L_3$$

$$L_1 \cap L_2 = L_2 \cap L_1$$

$$L_1 \cap L_1 = L_1$$

$$L_1 \cap \emptyset = \emptyset$$

$$L_1 \cdot (L_2 \cdot L_3) = (L_1 \cdot L_2) \cdot L_3$$

$$L_1 \cdot \{\varepsilon\} = L_1$$

$$\{\varepsilon\} \cdot L_1 = L_1$$

$$L_1 \cdot \emptyset = \emptyset$$

$$\emptyset \cdot L_1 = \emptyset$$

# Some Properties of Operations on Languages

$$L_1 \cdot (L_2 \cup L_3) = (L_1 \cdot L_2) \cup (L_1 \cdot L_3)$$

$$(L_1 \cup L_2) \cdot L_3 = (L_1 \cdot L_3) \cup (L_2 \cdot L_3)$$

$$(L_1^*)^* = L_1^*$$

$$\emptyset^* = \{\varepsilon\}$$

$$L_1^* = \{\varepsilon\} \cup (L_1 \cdot L_1^*)$$

$$L_1^* = \{\varepsilon\} \cup (L_1^* \cdot L_1)$$

$$(L_1 \cup L_2)^* = L_1^* \cdot (L_2 \cdot L_1^*)^*$$

$$(L_1 \cdot L_2)^R = L_2^R \cdot L_1^R$$