Name: doc. Ing. Zdeněk Sawa, Ph.D.
E-mail: zdenek.sawa@vsb.cz
Room: EA413
Web: http://www.cs.vsb.cz/sawa/tcs

On these pages you will find:
- Information about the course
- Slides from lectures
- Exercises for tutorials
- Recent news for the course
- A link to a page with animations
Requirements

- **Credit** (45 points):
  - A written test; it is necessary to obtain at least 20 points.

- **Exam** (55 points)
  - A written exam; it is necessary to obtain at least 6 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?
**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 $z = x + y$. 

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

### Example:
- **Input:** 8, 13, 3, 10, 1, 4  
- **Output:** 1, 3, 4, 8, 10, 13

### Remark:
A particular input of a problem is called an *instance* of the problem.
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 $v$.

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
Other Examples of Problems

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 \{Yes, 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?
Examples of 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?

$k = 3$
Examples of 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?

$k = 3$
Theoretical computer science overlaps with many other areas of mathematics and computer science:

- graph theory
- number theory
- computational geometry
- searching in text
- game theory
- ...
Formal Languages
An area of theoretical computer science dealing with questions concerning syntax.

- **Language** — a set of words
- **Word** — a sequence 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 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:**


Words over alphabet $\Sigma$: HELLO, XYZZY, COMPUTER
Alphabet and Word

Example 2:


A word over alphabet \( \Sigma_2 \): HELLO\(|\square\)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, 101010101010101010

Example 5:

Words over alphabet \( \Sigma_5 = \{a, b\} \): aababb, abbabbba, aaab
Example 6:

Alphabet $\Sigma_6$ is the set of all ASCII characters.

Example of a word:

```c
#include <stdio.h>

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

```c
#include <stdio.h> int main() {
    printf("Hello, world!\n");
    return 0;
}
```
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 $\Sigma$):

$$826, 13, 3901, 101, 128, 562$$

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

$$13, 101, 128, 562, 826, 3901$$

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

![Graph](image)

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 $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, , , (, )\}$. 
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.
To describe a language, there are several possibilities:

- We can enumerate all words of the language (however, this is possible only for small finite languages).

  **Example:** \( L = \{aab, babba, aaaaaa\} \)

- We can specify a property of the words of the language:

  **Example:** The language over alphabet \( \{0, 1\} \) containing all words with even number of occurrences of symbol 1.
In particular, the following two approaches are used in the theory of formal languages:

- To describe an (idealized) machine, device, algorithm, that recognizes words of the given language – approaches based on automata.

- To describe some mechanism that allows to generate all words of the given language – approaches based on grammars or regular expressions.
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”.
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 $w = abaab$ then $|w| = 5$.

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

For word $w = ababb$ we have $|w|_a = 2$ and $|w|_b = 3$.

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.

(Remark: In literature, sometimes $\lambda$ (lambda) is used to denote the empty word instead of $\varepsilon$.)

$$|\varepsilon| = 0$$
Concatenation of Words

One of operations we can do on words is the operation of concatenation: For example, the concatenation of words \texttt{OST} and \texttt{RAVA} is the word \texttt{OSTRAVA}.

The operation of concatenation is denoted by symbol \texttt{·} (similarly to multiplication). It is possible to omit this symbol.

\[
\texttt{OST} \cdot \texttt{RAVA} = \texttt{OSTRAVA}
\]

Concatenation is associative, i.e., for every three words \(u\), \(\nu\), and \(w\) we have

\[
(u \cdot \nu) \cdot w = u \cdot (\nu \cdot w)
\]

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 \((w_1 \cdot (w_2 \cdot w_3)) \cdot (w_4 \cdot w_5)\).
Concatenation of Words

Concatenation is not **commutative**, i.e., the following equality does not hold in general

\[ u \cdot v = v \cdot u \]

**Example:**

\[ \text{OST} \cdot \text{RAVA} \neq \text{RAVA} \cdot \text{OST} \]

It is obvious that the following holds for any words \( v \) and \( w \):

\[ |v \cdot w| = |v| + |w| \]

For every word \( w \) we also have:

\[ \varepsilon \cdot w = w \cdot \varepsilon = w \]
Definition

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

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

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

Example:

- Prefixes of the word $abaab$ are $\varepsilon$, $a$, $ab$, $aba$, $abaa$, $abaab$.
- Suffixes of the word $abaab$ are $\varepsilon$, $b$, $ab$, $aab$, $baab$, $abaab$.
- Subwords of the word $abaab$ are $\varepsilon$, $a$, $b$, $ab$, $ba$, $aa$, $aba$, $baa$, $aab$, $abaa$, $baab$, $abaab$. 
Order on Words

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

$$a_1 < a_2 < \ldots < a_n.$$ 

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

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

$$x <_L y \text{ iff:}$$

- $|x| < |y|$, 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

$$w_0, w_1, w_2, \ldots$$

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:

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

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 $<_L$. 
Finite Automata
Example: Consider words over alphabet \( \{0, 1\} \).

We would like to recognize a language \( L \) consisting of words with even number of symbols 1.

We want to design a device that reads a word and then tells us if the word belongs to the language \( L \) or not.
**Example:** Consider words over alphabet \( \{0, 1\} \).

We would like to recognize a language \( L \) consisting of words with even number of symbols \( 1 \).

We want to design a device that reads a word and then tells us if the word belongs to the language \( L \) or not.
Example: Consider words over alphabet \{0, 1\}.
We would like to recognize a language \(L\) consisting of words with even number of symbols 1.

We want to design a device that reads a word and then tells us if the word belongs to the language \(L\) or not.
**Example:** Consider words over alphabet \{0, 1\}.
We would like to recognize a language \( L \) consisting of words with even number of symbols 1.

We want to design a device that reads a word and then tells us if the word belongs to the language \( L \) or not.
Recognition of a Language

**Example:** Consider words over alphabet \( \{0, 1\} \). We would like to recognize a language \( L \) consisting of words with even number of symbols \( 1 \).

We want to design a device that reads a word and then tells us if the word belongs to the language \( L \) or not.
**Example:** Consider words over alphabet \( \{0, 1\} \).

We would like to recognize a language \( L \) consisting of words with even number of symbols 1.

We want to design a device that reads a word and then tells us if the word belongs to the language \( L \) or not.
Recognition of a Language

**Example:** Consider words over alphabet \( \{0, 1\} \).

We would like to recognize a language \( L \) consisting of words with even number of symbols 1.

We want to design a device that reads a word and then tells us if the word belongs to the language \( L \) or not.

```
0 1 0 1 1 1 0 1 0 0 1
```

Z. Sawa (TU Ostrava)  
Theoretical Computer Science  
September 13, 2019  34 / 490
**Example:** Consider words over alphabet \(\{0, 1\}\).

We would like to recognize a language \(L\) consisting of words with even number of symbols 1.

We want to design a device that reads a word and then tells us if the word belongs to the language \(L\) or not.
Recognition of a Language

**Example:** Consider words over alphabet \{0, 1\}. We would like to recognize a language \( L \) consisting of words with even number of symbols 1.

We want to design a device that reads a word and then tells us if the word belongs to the language \( L \) or not.

```
0 1 0 1 1 1 0 1 0 0 1
```
**Example:** Consider words over alphabet \{0, 1\}.

We would like to recognize a language $L$ consisting of words with even number of symbols 1.

We want to design a device that reads a word and then tells us if the word belongs to the language $L$ or not.
Example: Consider words over alphabet \( \{0, 1\} \).

We would like to recognize a language \( L \) consisting of words with even number of symbols 1.

We want to design a device that reads a word and then tells us if the word belongs to the language \( L \) or not.
Recognition of a Language

Example: Consider words over alphabet \( \{0, 1\} \).

We would like to recognize a language \( L \) consisting of words with even number of symbols \( 1 \).

We want to design a device that reads a word and then tells us if the word belongs to the language \( L \) or not.

\[
\begin{array}{cccccccccc}
0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
\end{array}
\]

YES
The first idea: To count the number of occurrences of symbol 1.
The first idea: To count the number of occurrences of symbol 1.
The first idea: To count the number of occurrences of symbol 1.
The first idea: To count the number of occurrences of symbol 1.
The first idea: To count the number of occurrences of symbol 1.
The first idea: To count the number of occurrences of symbol 1.
The first idea: To count the number of occurrences of symbol 1.
The first idea: To count the number of occurrences of symbol 1.
The first idea: To count the number of occurrences of symbol 1.
The first idea: To count the number of occurrences of symbol 1.
The first idea: To count the number of occurrences of symbol 1.
The first idea: To count the number of occurrences of symbol 1.

YES – 6 is an even number
The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).
The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).
The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).
The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).

\[ \begin{array}{cccccccc}
0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{array} \]
The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).
The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).
The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).
**The second idea:** In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).
The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).
The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).
The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).
The second idea: In fact, we just need to remember if the number of symbols 1 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).
Recognition of a Language

The behaviour of the device can be described by the following graph:

\[ \begin{array}{cc}
E & O \\
\end{array} \]
The behaviour of the device can be described by the following graph:
Recognition of a Language

The behaviour of the device can be described by the following graph:
Recognition of a Language

The behaviour of the device can be described by the following graph:

![Graph](image-url)
Recognition of a Language

The behaviour of the device can be described by the following graph:
Recognition of a Language

The behaviour of the device can be described by the following graph:

![Graph Image](image-url)
The behaviour of the device can be described by the following graph:
Recognition of a Language

The behaviour of the device can be described by the following graph:
Recognition of a Language

The behaviour of the device can be described by the following graph:

![Graph Description]

The behaviour of the device can be described by the following graph:
The behaviour of the device can be described by the following graph:
Recognition of a Language

The behaviour of the device can be described by the following graph:

![Graph Diagram]

0 1 0 1 1 1 0 1 0 0 1
Recognition of a Language

The behaviour of the device can be described by the following graph:

\[ E \]
Recognition of a Language

The behaviour of the device can be described by the following graph:

![Graph diagram]

0 1 0 1 1 1 0 1 0 0 1

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019 37 / 490
The behaviour of the device can be described by the following graph:
The behaviour of the device can be described by the following graph:
Recognition of a Language

The behaviour of the device can be described by the following graph:
The behaviour of the device can be described by the following graph:

![Graph](attachment:graph.png)
Recognition of a Language

The behaviour of the device can be described by the following graph:
The behaviour of the device can be described by the following graph:
A deterministic finite automaton consists of states and transitions. One of the states is denoted as an initial state and some of states are denoted as accepting.
Deterministic Finite Automaton

Formally, a **deterministic finite automaton (DFA)** is defined as a tuple

\[(Q, \Sigma, \delta, q_0, F)\]

where:

- \(Q\) is a nonempty finite set of **states**
- \(\Sigma\) is an **alphabet** (a nonempty finite set of symbols)
- \(\delta : Q \times \Sigma \rightarrow Q\) is a **transition function**
- \(q_0 \in Q\) is an **initial state**
- \(F \subseteq Q\) is a set of **accepting states**
Deterministic Finite Automaton

\[ Q = \{1, 2, 3, 4, 5\} \]
\[ \Sigma = \{a, b\} \]
\[ q_0 = 1 \]
\[ F = \{1, 4, 5\} \]

\[ \delta(1, a) = 2 \quad \delta(1, b) = 1 \]
\[ \delta(2, a) = 4 \quad \delta(2, b) = 5 \]
\[ \delta(3, a) = 1 \quad \delta(3, b) = 4 \]
\[ \delta(4, a) = 1 \quad \delta(4, b) = 3 \]
\[ \delta(5, a) = 4 \quad \delta(5, b) = 5 \]
Instead of

$$
\begin{align*}
\delta(1, a) &= 2 & \delta(1, b) &= 1 \\
\delta(2, a) &= 4 & \delta(2, b) &= 5 \\
\delta(3, a) &= 1 & \delta(3, b) &= 4 \\
\delta(4, a) &= 1 & \delta(4, b) &= 3 \\
\delta(5, a) &= 4 & \delta(5, b) &= 5
\end{align*}
$$

we rather use a more succinct representation as a table or a depicted graph:

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leftrightarrow$</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>$\leftrightarrow$</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>$\leftrightarrow$</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Deterministic Finite Automaton
Deterministic Finite Automaton

Z. Sawa (TU Ostrava)
Theoretical Computer Science
September 13, 2019
Deterministic Finite Automaton

Z. Sawa (TU Ostrava)
Deterministic Finite Automaton

Definition

Let us have a DFA $A = (Q, \Sigma, \delta, q_0, F)$.

By $q \xrightarrow{w} q'$, where $q, q' \in Q$ and $w \in \Sigma^*$, we denote the fact that the automaton, starting in state $q$ goes to state $q'$ by reading word $w$.

Remark: $\xrightarrow{} \subseteq Q \times \Sigma^* \times Q$ is a ternary relation.

Instead of $(q, w, q') \in \xrightarrow{}$ we write $q \xrightarrow{w} q'$.

It holds for a DFA that for each state $q$ and each word $w$ there is exactly one state $q'$ such that $q \xrightarrow{w} q'$. 
Relation $\rightarrow$ can be formally defined by the following inductive definition:

- $q \xrightarrow{\varepsilon} q$ for each $q \in Q$
- For $a \in \Sigma$ and $w \in \Sigma^*$:
  
  $q \xrightarrow{aw} q'$ iff there is $q'' \in Q$ such that $\delta(q, a) = q''$ and $q'' \xrightarrow{w} q'$.
A word $w \in \Sigma^*$ is **accepted** by a deterministic finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ iff there exists a state $q \in F$ such that $q_0 \xrightarrow{w} q$.

**Definition**

A **language** accepted by a given deterministic finite automaton $A = (Q, \Sigma, \delta, q_0, F)$, denoted $L(A)$, is the set of all words accepted by the automaton, i.e.,

$$L(A) = \{w \in \Sigma^* \mid \exists q \in F : q_0 \xrightarrow{w} q\}$$
A language $L$ is **regular** iff there exists some deterministic finite automaton accepting $L$, i.e., DFA $A$ such that $L(A) = L$. 

**Definition**

A language $L$ is regular iff there exists some deterministic finite automaton accepting $L$, i.e., DFA $A$ such that $L(A) = L$. 

Z. Sawa (TU Ostrava) Theoretical Computer Science September 13, 2019 46 / 490
All 3 automata accept the language of all words with an even number of a’s.
Equivalence of Automata

**Definition**

We say automata $A_1, A_2$ are **equivalent** if $L(A_1) = L(A_2)$. 
The automaton accepts the language
\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains subword } ab \} \]

There is no input sequence such that after reading it, the automaton gets to states 3, 4, or 5.
The automaton accepts the language
\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains subword } ab \} \]

There is no input sequence such that after reading it, the automaton gets to states 3, 4, or 5.

If we remove these states, the automaton still accepts the same language \( L \).
Unreachable States of an Automaton

Definition

A state \( q \) of a finite automaton \( A = (Q, \Sigma, \delta, q_0, F) \) is reachable if there exists a word \( w \) such that \( q_0 \xrightarrow{w} q \).

Otherwise the state is unreachable.

- There is no path in a graph of an automaton going from the initial state to some unreachable state.
- Unreachable states can be removed from an automaton (together with all transitions going to them and from them). The language accepted by the automaton is not affected.
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:** It is assumed the languages involved in these operations use the same alphabet $\Sigma$. 
Set Operations on Languages

Formally:

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

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

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

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

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

Example:

$L_1 = \{abb, ba\}$

$L_2 = \{a, ab, bbb\}$

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

$abba \quad abbab \quad abbbbb \quad baa \quad baab \quad babb$
Iteration of a Language

**Definition**

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

I.e. $w \in L^*$ iff

\[
\exists n \in \mathbb{N} : \exists w_1, w_2, \ldots, w_n \in L : w = w_1 w_2 \cdots w_n
\]

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

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

**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 – Alternative Definition

At first, for a language \( L \) and a number \( k \in \mathbb{N} \) we define the language \( L^k \):

\[
L^0 = \{ \varepsilon \}, \quad L^k = L^{k-1} \cdot L \quad \text{for } k \geq 1
\]

This means

\[
\begin{align*}
L^0 &= \{ \varepsilon \} \\
L^1 &= L \\
L^2 &= L \cdot L \\
L^3 &= L \cdot L \cdot L \\
L^4 &= L \cdot L \cdot L \cdot L \\
L^5 &= L \cdot L \cdot L \cdot L \cdot L \\
&\vdots
\end{align*}
\]

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

\[
\text{aaaaaa} \quad \text{aaaab} \quad \text{aabaa} \quad \text{aabb} \quad \text{baaaa} \quad \text{baab} \quad \text{bbaa} \quad \text{bbb}
\]
The **iteration (Kleene star)** of language $L$ is the language

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

**Remark:**

$$\bigcup_{k \geq 0} L^k = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots$$
Remark: Sometimes, notation $L^+$ is used as an abbreviation for $L \cdot L^*$, i.e.,

$$L^+ = \bigcup_{k \geq 1} L^k$$
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:** $w = \text{HELLO}$ $w^R = \text{OLLEH}$

Formally, for $w = a_1a_2\cdots a_n$ (where $a_i \in \Sigma$) is $w^R = a_na_{n-1}\cdots a_1$. 
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: \(L = \{ab, baaba, aaab\}\)
\(L^R = \{ba, abaab, baaa\}\)
Let us have the following two automata:

Do both of them accept the word $ababb$?
Let us have the following two automata:

Do both of them accept the word $ababb$?
An Automaton for Intersection of Languages

Let us have the following two automata:

\[ \begin{align*}
\text{State } 0 & \quad \text{Label } a, b \\
\text{State } 1 & \quad \text{Label } a, b \\
\text{State } 2 & \quad \text{Label } a \\
\text{State } 3 & \quad \text{Label } a, b
\end{align*} \]

\[ \begin{align*}
\text{State } A & \quad \text{Label } a, b \\
\text{State } B & \quad \text{Label } b
\end{align*} \]

Do both of them accept the word \( ababb \)?
Let us have the following two automata:

[Diagram of two automata]

Do both of them accept the word $ababb$?
Let us have the following two automata:

Do both of them accept the word $aba$?
Let us have the following two automata:

![Automata Diagram]

Do both of them accept the word \textit{ababb}?
Let us have the following two automata:

![Automata Diagram]

Do both of them accept the word $ababb$?
An Automaton for Intersection of Languages

[Diagram of an automaton with states and transitions labeled with symbols a and b.]

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019  62 / 490
An Automaton for Intersection of Languages
An Automaton for Intersection of Languages

Z. Sawa (TU Ostrava)
Theoretical Computer Science
September 13, 2019 62 / 490
An Automaton for Intersection of Languages

Z. Sawa (TU Ostrava)

Theoretical Computer Science

September 13, 2019 62 / 490
An Automaton for Intersection of Languages

\[ \begin{array}{c}
\text{0} & \text{a} & \text{1} & \text{b} & \text{2} & \text{a} & \text{3} & \text{a, b} \\
\text{b} & \text{a} & \text{a} & \text{b} & \text{a} & \text{b} & \text{b} \\
\end{array} \]
An Automaton for Intersection of Languages

Z. Sawa (TU Ostrava)
An Automaton for Intersection of Languages

[Diagram of automata with states and transitions labeled with 'a' and 'b']

Z. Sawa (TU Ostrava)
An Automaton for Intersection of Languages
An Automaton for Intersection of Languages
An Automaton for Intersection of Languages
An Automaton for Intersection of Languages
An Automaton for Intersection of Languages
An Automaton for Intersection of Languages
An Automaton for Intersection of Languages

Z. Sawa (TU Ostrava)
An Automaton for Intersection of Languages
An Automaton for Intersection of Languages

[Diagram of automata with states and transitions]

Z. Sawa (TU Ostrava)
An Automaton for Intersection of Languages

![Automaton Diagram]

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019  62 / 490
An Automaton for Intersection of Languages
An Automaton for Intersection of Languages

[Diagram showing automata for the intersection of languages]
An Automaton for Intersection of Languages
Formally, the construction can be described as follows:

We assume we have two deterministic finite automata $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$.

We construct DFA $A = (Q, \Sigma, \delta, q_0, F)$ where:

- $Q = Q_1 \times Q_2$
- $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$ for each $q_1 \in Q_1$, $q_2 \in Q_2$, $a \in \Sigma$
- $q_0 = (q_{01}, q_{02})$
- $F = F_1 \times F_2$

It is not difficult to check that for each word $w \in \Sigma^*$ we have $w \in L(A)$ iff $w \in L(A_1)$ and $w \in L(A_2)$, i.e.,

$$L(A) = L(A_1) \cap L(A_2)$$
**Theorem**

If languages $L_1, L_2 \subseteq \Sigma^*$ are regular then also the language $L_1 \cap L_2$ is regular.

**Proof:** Let us assume that $A_1$ and $A_2$ are deterministic finite automata such that

$$L_1 = L(A_1) \quad \quad L_2 = L(A_2)$$

Using the described construction, we can construct a deterministic finite automaton $A$ such that

$$L(A) = L(A_1) \cap L(A_2) = L_1 \cap L_2$$
An Automaton for the Union of Languages

Z. Sawa (TU Ostrava)
An Automaton for the Union of Languages

Z. Sawa (TU Ostrava)
An Automaton for the Union of Languages

Z. Sawa (TU Ostrava)
An Automaton for the Union of Languages

Theoretical Computer Science

September 13, 2019 65 / 490
An Automaton for the Union of Languages
The construction of an automaton $A$ that accepts the union of languages accepted by automata $A_1$ and $A_2$, i.e., the language

$$L(A_1) \cup L(A_1)$$

is almost identical as in the case of the automaton accepting $L(A_1) \cap L(A_2)$.

The only difference is the set of accepting states:

$$F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$$
The construction of an automaton $A$ that accepts the union of languages accepted by automata $A_1$ and $A_2$, i.e., the language

$$L(A_1) \cup L(A_1)$$

is almost identical as in the case of the automaton accepting $L(A_1) \cap L(A_2)$.

The only difference is the set of accepting states:

- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

**Theorem**

If languages $L_1, L_2 \subseteq \Sigma^*$ are regular then also the language $L_1 \cup L_2$ is regular.
An Automaton for the Complement of a Language
An Automaton for the Complement of a Language

The diagram shows an automaton with states labeled 0 through 3. The transitions are labeled with symbols 'a' and 'b'. The automaton transitions between states as follows:

- From state 0, on 'a' it moves to state 1, and on 'b' it moves to state 2.
- From state 1, on 'a' it moves to state 2, and on 'b' it moves to state 3.
- From state 2, on 'a' it moves to state 3.
- From state 3, on 'a' it moves to state 0, and on 'b' it moves to state 1.

The states 0, 1, 2, and 3 are connected in a loop, with 'a' and 'b' as inputs to move between states.
Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ we construct DFA $A' = (Q, \Sigma, \delta, q_0, Q - F)$.

It is obvious that for each word $w \in \Sigma^*$ we have $w \in L(A')$ iff $w \notin L(A)$, i.e.,

$$L(A') = \overline{L(A)}$$
Complement of a Regular Language

Given a DFA \( A = (Q, \Sigma, \delta, q_0, F) \) we construct DFA \( A' = (Q, \Sigma, \delta, q_0, Q - F) \).

It is obvious that for each word \( w \in \Sigma^* \) we have \( w \in L(A') \) iff \( w \not\in L(A) \), i.e.,

\[
L(A') = \overline{L(A)}
\]

**Theorem**

If a language \( L \) is regular then also its complement \( \overline{L} \) is regular.
The number of transitions going from one state and labelled with the same symbol can be arbitrary (including zero).

There can be more than one initial state in the automaton.
Nondeterministic Finite Automaton

The diagram represents a nondeterministic finite automaton (NFA) with states labeled 1 to 5 and transitions labeled with symbols a and b. The automaton starts at state 1 and has transitions:

- From state 1, if input is a, it can go to state 2.
- From state 1, if input is b, it remains at state 1.
- From state 2, if input is a, it goes to state 3.
- From state 2, if input is b, it goes to state 4.
- From state 3, if input is b, it goes to state 2.
- From state 4, if input is b, it goes back to state 1.
- From state 4, if input is a, it goes to state 5.
- From state 5, if input is b, it goes back to state 1.
- A loop on state 1 allows transitioning back to itself on input b.
Nondeterministic Finite Automaton

Theoretical Computer Science

Z. Sawa (TU Ostrava)
Nondeterministic Finite Automaton
Nondeterministic Finite Automaton

Theoretical Computer Science

Z. Sawa (TU Ostrava)

September 13, 2019
Nondeterministic Finite Automaton

1 → 3 → 4 → 2 → 5

a b a b b
Nondeterministic Finite Automaton

Diagram:

1. a → 3
2. b → 4
3. a → 2
4. b → 5
5. b → 5

Transition Table:

<table>
<thead>
<tr>
<th>States</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Nondeterministic Finite Automaton

Z. Sawa (TU Ostrava)
Nondeterministic Finite Automaton

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019  70 / 490
Nondeterministic Finite Automaton

Z. Sawa (TU Ostrava)
Theoretical Computer Science
September 13, 2019 70 / 490
A nondeterministic finite automaton accepts a given word if there exists at least one computation of the automaton that accepts the word.
A nondeterministic finite automaton accepts a given word if there exists at least one computation of the automaton that accepts the word.
Example: A forest representing all possible computations over the word $abb$. 
Formally, a **nondeterministic finite automaton (NFA)** is defined as a tuple

$$(Q, \Sigma, \delta, I, F)$$

where:

- $Q$ is a finite set of **states**
- $\Sigma$ is a finite **alphabet**
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ is a **transition function**
- $I \subseteq Q$ is a set of **initial states**
- $F \subseteq Q$ is a set of **accepting states**
Transformation of NFA to DFA
Transformation of NFA to DFA

Z. Sawa (TU Ostrava)
Theoretical Computer Science
September 13, 2019 74 / 490
Transformation of NFA to DFA
Transformation of NFA to DFA
Transformation of NFA to DFA

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019  74 / 490
Transformation of NFA to DFA

The diagram shows a non-deterministic finite automaton (NFA) with states numbered 1, 2, and 3. The transitions are labeled with symbols a, b, and a,b.

The states are connected as follows:
- State 1 transitions to state 2 on input b, to state 3 on input a.
- State 2 transitions to state 1 on input a and to state 3 on input b.
- State 3 transitions to state 2 on input a and to state 1 on input b.

The input alphabet is a, b, and a,b.

The diagram also includes a box with the sequence a a b b a, indicating the input to the automaton.
Transformation of NFA to DFA

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019 75 / 490
Transformation of NFA to DFA

Theoretical Computer Science
September 13, 2019
Transformation of NFA to DFA

Z. Sawa (TU Ostrava)
Transformation of NFA to DFA

Theoretical Computer Science

September 13, 2019 75 / 490
Transformation of NFA to DFA

Theoretical Computer Science

September 13, 2019 75 / 490
Transformation of NFA to DFA

Z. Sawa (TU Ostrava)
Transformation of NFA to DFA
Transformation of NFA to DFA
Transformation of NFA to DFA

Theoretical Computer Science

September 13, 2019 75 / 490
Transformation of NFA to DFA
Transformation of NFA to DFA

Z. Sawa (TU Ostrava)
Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leftrightarrow 1$</td>
<td>$-$</td>
<td>2, 3</td>
</tr>
<tr>
<td>$\rightarrow 2$</td>
<td>2, 3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>$-$</td>
</tr>
</tbody>
</table>
### Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>←</td>
<td>1</td>
<td>–</td>
<td>2, 3</td>
</tr>
<tr>
<td>→</td>
<td>2</td>
<td>2, 3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>–</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
</table>
### Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leftrightarrow 1$</td>
<td>$-$</td>
<td>2, 3</td>
</tr>
<tr>
<td>$\rightarrow 2$</td>
<td>2, 3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>$-$</td>
</tr>
</tbody>
</table>

$\leftrightarrow \{1, 2\}$

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leftrightarrow {1, 2}$</td>
<td>$\rightarrow 2$</td>
</tr>
<tr>
<td>2, 3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>$-$</td>
</tr>
</tbody>
</table>
### Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>2, 3</td>
</tr>
<tr>
<td>2</td>
<td>2, 3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

↔ \{1, 2\} ↔ \{2, 3\}
### Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>⇐ 1</td>
<td></td>
<td>2, 3</td>
<td></td>
</tr>
<tr>
<td>→ 2</td>
<td>2, 3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>⇐ {1, 2}</td>
<td>{2, 3}</td>
<td></td>
</tr>
<tr>
<td>{2, 3}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leftrightarrow 1$</td>
<td>$-$</td>
<td>$2, 3$</td>
</tr>
<tr>
<td>$\rightarrow 2$</td>
<td>$2, 3$</td>
<td>$3$</td>
</tr>
<tr>
<td>$3$</td>
<td>$1$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leftrightarrow {1, 2}$</td>
<td>${2, 3}$</td>
<td>${2, 3}$</td>
</tr>
<tr>
<td>${2, 3}$</td>
<td>${2, 3}$</td>
<td>${2, 3}$</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava)

Theoretical Computer Science

September 13, 2019 76 / 490
### Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1, 2}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>2</td>
<td>{2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>3</td>
<td>{1, 2, 3}</td>
<td>{1, 2, 3}</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava)
Transformation of NFA to DFA

\[
\begin{array}{c|cc}
& a & b \\
\hline
\leftrightarrow 1 & \rightarrow 2 & \leftrightarrow 1 \\
\rightarrow 2 & 2, 3 & 3 \\
3 & 1 & \rightarrow 2
\end{array}
\]
### Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2, 3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2, 3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,3</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>2,3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 2}</td>
<td>{2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>{2, 3}</td>
<td>{1, 2, 3}</td>
<td>{3}</td>
</tr>
<tr>
<td>{1, 2, 3}</td>
<td>{1, 2, 3}</td>
<td>{3}</td>
</tr>
</tbody>
</table>
### Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \leftrightarrow 1 )</td>
<td>(-)</td>
<td>( 2, 3 )</td>
</tr>
<tr>
<td>( \rightarrow 2 )</td>
<td>( 2, 3 )</td>
<td>( 3 )</td>
</tr>
<tr>
<td>( 3 )</td>
<td>( 1 )</td>
<td>(-)</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
\leftrightarrow \{1, 2\} & \quad \{2, 3\} \\
\{2, 3\} & \quad \{1, 2, 3\} \\
\leftarrow \{1, 2, 3\} & \quad \{1, 2, 3\} \\
\{3\} & \quad \{3\}
\end{align*} \]
Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2,3</td>
</tr>
<tr>
<td>2</td>
<td>2,3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 2}</td>
<td>{2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>{2, 3}</td>
<td>{1, 2, 3}</td>
<td>{3}</td>
</tr>
<tr>
<td>{1, 2, 3}</td>
<td>{1, 2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>{3}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leftrightarrow 1$</td>
<td>$\rightarrow 2$</td>
<td>$\rightarrow 3$</td>
</tr>
<tr>
<td>$\rightarrow 2$</td>
<td>$2, 3$</td>
<td>$3$</td>
</tr>
<tr>
<td>$\rightarrow 3$</td>
<td>$1$</td>
<td>$\rightarrow 1$</td>
</tr>
</tbody>
</table>

| $\leftrightarrow \{1, 2\}$ | $\leftrightarrow \{1, 2, 3\}$ | $\leftrightarrow \{2, 3\}$ |
| $\leftrightarrow \{2, 3\}$ | $\rightarrow \{1, 2, 3\}$ | $\rightarrow \{2, 3\}$ |
| $\leftrightarrow \{2, 3\}$ | $\rightarrow \{1, 2, 3\}$ | $\rightarrow \{3\}$ |
| $\leftarrow \{1, 2, 3\}$ | $\leftarrow \{1, 2, 3\}$ | $\leftarrow \{2, 3\}$ |
| $\leftarrow \{3\}$ | $\leftarrow \{1\}$ | $\leftarrow \{1\}$ |
| $\leftarrow \{1\}$ |
Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 3</td>
<td>←</td>
</tr>
<tr>
<td>2</td>
<td>2, 3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>←</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 2}</td>
<td>{2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>{2, 3}</td>
<td>{1, 2, 3}</td>
<td>{3}</td>
</tr>
<tr>
<td>{1, 2, 3}</td>
<td>{1, 2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>{3}</td>
<td>{1}</td>
<td>⌀</td>
</tr>
<tr>
<td>{1}</td>
<td>←</td>
<td></td>
</tr>
</tbody>
</table>
Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2, 3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 2}</td>
<td>{2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>{2, 3}</td>
<td>{1, 2, 3}</td>
<td>{3}</td>
</tr>
<tr>
<td>{1, 2, 3}</td>
<td>{1, 2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>{3}</td>
<td>{1}</td>
<td>{}\</td>
</tr>
<tr>
<td>{1}</td>
<td>{}\</td>
<td>{}\</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019  76 / 490
### Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>2, 3</td>
</tr>
<tr>
<td>2</td>
<td>2, 3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 2}</td>
<td>{2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>{2, 3}</td>
<td>{1, 2, 3}</td>
<td>{3}</td>
</tr>
<tr>
<td>{1, 2, 3}</td>
<td>{1, 2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>{3}</td>
<td>{1}</td>
<td>∅</td>
</tr>
<tr>
<td>{1}</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava)
### Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>−</td>
<td>2, 3</td>
</tr>
<tr>
<td>2</td>
<td>2, 3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>−</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1, 2}</td>
<td>{2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>{2, 3}</td>
<td>{1, 2, 3}</td>
<td>{3}</td>
</tr>
<tr>
<td>{1, 2, 3}</td>
<td>{1, 2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>{3}</td>
<td>{1}</td>
<td>∅</td>
</tr>
<tr>
<td>{1}</td>
<td>∅</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>
### Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="https://via.placeholder.com/150" alt="Alternative state" /></td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
</tr>
<tr>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
</tr>
<tr>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
</tr>
<tr>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
</tr>
<tr>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
<td><img src="https://via.placeholder.com/150" alt="States" /></td>
</tr>
</tbody>
</table>
### Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2, 3</td>
</tr>
<tr>
<td>2</td>
<td>2, 3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The table above represents the transformation from an NFA to a DFA. Each state in the NFA is mapped to a set of states in the DFA, and each transition in the NFA is mapped to a transition in the DFA with the same input symbol. The table shows the corresponding states and transitions for the input symbols 'a' and 'b'.
Remark: When a nondeterministic automaton with $n$ states is transformed into a deterministic one, the resulting automaton can have $2^n$ states.

For example when we transform an automaton with 20 states, the resulting automaton can have $2^{20} = 1048576$ states.

It is often the case that the resulting automaton has far less than $2^n$ states. However, the worst cases are possible.
Generalized Nondeterministic Finite Automaton

Theoretical Computer Science
September 13, 2019 78 / 490
Generalized Nondeterministic Finite Automaton

Z. Sawa (TU Ostrava)
Generalized Nondeterministic Finite Automaton

Z. Sawa (TU Ostrava)
Theoretical Computer Science
September 13, 2019 78 / 490
Generalized Nondeterministic Finite Automaton

Theoretical Computer Science
September 13, 2019 78 / 490
Generalized Nondeterministic Finite Automaton

1
2
3
4
5

1 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow 5

Z. Sawa (TU Ostrava) Theoretical Computer Science September 13, 2019 78 / 490
Generalized Nondeterministic Finite Automaton

Theoretical Computer Science

Z. Sawa (TU Ostrava)
Compared to a nondeterministic finite automaton, a generalized nondeterministic finite automaton has the so called $\varepsilon$-transitions, i.e., transitions labelled with symbol $\varepsilon$.

When $\varepsilon$-transition is performed, only the state of the control unit is changed but the head on the tape is not moved.

Remark: The computations of a generalized nondeterministic automaton can be of an arbitrary length, even infinite (if the graph of the automaton contains a cycle consisting only of $\varepsilon$-transitions) regardless of the length of the word on the tape.
Formally, a generalized nondeterministic finite automaton (GNFA) is defined as a tuple

\[(Q, \Sigma, \delta, I, F)\]

where:

- \(Q\) is a finite set of states
- \(\Sigma\) is a finite alphabet
- \(\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)\) is a transition function
- \(I \subseteq Q\) is a set of initial states
- \(F \subseteq Q\) is a set of accepting states

**Remark:** NFA can be viewed as a special case of GNFA, where \(\delta(q, \varepsilon) = \emptyset\) for all \(q \in Q\).
Transformation to a Deterministic Finite Automaton

A generalized nondeterministic finite automaton can be transformed into a deterministic one using a similar construction as a nondeterministic finite automaton with the difference that we add to sets of states also all states that are reachable from already added states by some sequence of $\varepsilon$-transitions.
The diagram represents a finite state automaton with states labeled 1, 2, 3, and transitions labeled with symbols a, b, and ε. The states and transitions are as follows:

- State 1 has transitions labeled a, b, and ε to states 2, 2, and 3, respectively.
- State 2 has a transition labeled a to state 3.
- State 3 has a transition labeled a, b to state 2 and a transition labeled a to state 1.

The diagram illustrates the paths and transitions within the automaton, with specific sets of symbols associated with each transition.
Transformation of GNFA to DFA

Before formally describing the transition of GNFA to DFA, let us introduce some auxiliary definitions.

Let us assume some given GNFA $A = (Q, \Sigma, \delta, I, F)$.

Let us define the function $\hat{\delta} : \mathcal{P}(Q) \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$ so that for $K \subseteq Q$ and $a \in \Sigma \cup \{\varepsilon\}$ there is

$$\hat{\delta}(K, a) = \bigcup_{q \in K} \delta(q, a)$$
Transformation of GNFA to DFA

For $K \subseteq Q$, let $\text{Cl}_\varepsilon(K)$ be the all states reachable from the states from the set $K$ by some arbitrary sequence of $\varepsilon$-transitions.

This means that the function $\text{Cl}_\varepsilon : \mathcal{P}(Q) \to \mathcal{P}(Q)$ is defined so that for $K \subseteq Q$ is $\text{Cl}_\varepsilon(K)$ the smallest (with respect to inclusion) set satisfying the following two conditions:

- $K \subseteq \text{Cl}_\varepsilon(K)$
- For each $q \in \text{Cl}_\varepsilon(K)$ it holds that $\delta(q, \varepsilon) \subseteq \text{Cl}_\varepsilon(K)$.

**Remark:** Let us note that $\text{Cl}_\varepsilon(\text{Cl}_\varepsilon(K)) = \text{Cl}_\varepsilon(K)$ for arbitrary $K$.

Let us also note that in the case of NFA (where $\delta(q, \varepsilon) = \emptyset$ for each $q \in Q$) is $\text{Cl}_\varepsilon(K) = K$. 
Transformation of GNFA to DFA

For a given GNFA $A = (Q, \Sigma, \delta, I, F)$ we can now construct DFA $A' = (Q', \Sigma, \delta', q'_0, F')$, where:

- $Q' = \mathcal{P}(Q)$ (so $K \in Q'$ means that $K \subseteq Q$)
- $\delta': Q' \times \Sigma \rightarrow Q'$ is defined so that for $K \in Q'$ and $a \in \Sigma$:
  \[ \delta'(K, a) = \text{Cl}_\varepsilon(\hat{\delta}(\text{Cl}_\varepsilon(K), a)) \]

- $q'_0 = \text{Cl}_\varepsilon(I)$
- $F' = \{K \in Q' : \text{Cl}_\varepsilon(K) \cap F \neq \emptyset\}$

It is not difficult to verify that $L(A) = L(A')$. 
\[ \Sigma = \{a, b, c, d\} \]
Concatenation of Languages

\[ \Sigma = \{a, b, c, d\} \]

\[ A_1: \]
\[ A_2: \]

\[ A: \]

\[ L(A) = L(A_1) \cdot L(A_2) \]
\[ \Sigma = \{a, b, c, d\} \]

**An incorrect construction:**

\[ acdbac \in L(A) \text{ but } acdbac \notin L(A_1) \cdot L(A_2) \]
Concatenation of Languages

$A_1$  

$A_2$
Concatenation of Languages

\[ A_1 \] 

\[ A_2 \] 

\[ A \]
Iteration of a Language

\[ A_1 \]
Iteration of a Language

\[ A_1 \]

\[ A \]

Z. Sawa (TU Ostrava)
Theoretical Computer Science
September 13, 2019 88 / 490
An alternative construction for the union of languages:
An alternative construction for the union of languages:
The set of (all) regular languages is closed with respect to:

- union
- intersection
- complement
- concatenation
- iteration
- ...
Regular Expressions
Regular Expressions

Regular expressions describing languages over an alphabet $\Sigma$:

- $\emptyset, \varepsilon, a$ (where $a \in \Sigma$) are regular expressions:
  - $\emptyset \ldots$ denotes the empty language
  - $\varepsilon \ldots$ denotes the language $\{\varepsilon\}$
  - $a \ldots$ denotes the language $\{a\}$

- If $\alpha, \beta$ are regular expressions then also $(\alpha + \beta), (\alpha \cdot \beta), (\alpha^*)$ are regular expressions:
  - $(\alpha + \beta) \ldots$ denotes the union of languages denoted $\alpha$ and $\beta$
  - $(\alpha \cdot \beta) \ldots$ denotes the concatenation of languages denoted $\alpha$ and $\beta$
  - $(\alpha^*) \ldots$ denotes the iteration of a language denoted $\alpha$

- There are no other regular expressions except those defined in the two points mentioned above.
Example: alphabet $\Sigma = \{0, 1\}$

- According to the definition, 0 and 1 are regular expressions.
Regular Expressions

Example: alphabet $\Sigma = \{0, 1\}$

- According to the definition, $0$ and $1$ are regular expressions.
- Since $0$ and $1$ are regular expression, $(0 + 1)$ is also a regular expression.
Example: alphabet $\Sigma = \{0, 1\}$

- According to the definition, 0 and 1 are regular expressions.
- Since 0 and 1 are regular expression, $(0 + 1)$ is also a regular expression.
- Since 0 is a regular expression, $(0^*)$ is also a regular expression.
Example: alphabet $\Sigma = \{0, 1\}$

- According to the definition, 0 and 1 are regular expressions.
- Since 0 and 1 are regular expression, $(0 + 1)$ is also a regular expression.
- Since 0 is a regular expression, $(0^*)$ is also a regular expression.
- Since $(0 + 1)$ and $(0^*)$ are regular expressions, $((0 + 1) \cdot (0^*))$ is also a regular expression.
Example: alphabet $\Sigma = \{0, 1\}$

- According to the definition, 0 and 1 are regular expressions.
- Since 0 and 1 are regular expression, $(0 + 1)$ is also a regular expression.
- Since 0 is a regular expression, $(0^*)$ is also a regular expression.
- Since $(0 + 1)$ and $(0^*)$ are regular expressions, $((0 + 1) \cdot (0^*))$ is also a regular expression.

Remark: If $\alpha$ is a regular expression, by $[\alpha]$ we denote the language defined by the regular expression $\alpha$.

$$[[(0 + 1) \cdot (0^*)]] = \{0, 1, 00, 10, 000, 100, 0000, 1000, 00000, \ldots\}$$
The structure of a regular expression can be represented by an abstract syntax tree:

$$(((0 \cdot 1)^* \cdot 1) \cdot (1 \cdot 1)) + (((0 \cdot 0) + 1)^*)$$
The formal definition of semantics of regular expressions:

- $[\emptyset] = \emptyset$
- $[\varepsilon] = \{\varepsilon\}$
- $[a] = \{a\}$
- $[\alpha^*] = [\alpha]^*$
- $[\alpha \cdot \beta] = [\alpha] \cdot [\beta]$  
- $[\alpha + \beta] = [\alpha] \cup [\beta]$
Regular Expressions

To make regular expressions more lucid and succinct, we use the following conventions:

- The outward pair of parentheses can be omitted.
- We can omit parentheses that are superflous due to associativity of operations of union (+) and concatenation (·).
- We can omit parentheses that are superflous due to the defined priority of operators (iteration (*) has the highest priority, concatenation (·) has lower priority, and union (+) has the lowest priority).
- A dot denoting concatenation can be omitted.

Example: Instead of

\[ (((((0 \cdot 1)^*) \cdot 1) \cdot (1 \cdot 1)) + (((0 \cdot 0) + 1)^*)) \]

we usually write

\[ (01)^*111 + (00 + 1)^* \]
Examples: In all examples $\Sigma = \{0, 1\}$.

$0 \ldots$ the language containing the only word $0$
Examples: In all examples $\Sigma = \{0, 1\}$.

- $0 \ldots$ the language containing the only word $0$
- $01 \ldots$ the language containing the only word $01$
Examples: In all examples $\Sigma = \{0, 1\}$.

0 . . . the language containing the only word 0

01 . . . the language containing the only word 01

0 + 1 . . . the language containing two words 0 and 1
Examples: In all examples $\Sigma = \{0, 1\}$.

- $0 \ldots$ the language containing the only word 0
- $01 \ldots$ the language containing the only word 01
- $0 + 1 \ldots$ the language containing two words 0 and 1
- $0^* \ldots$ the language containing words $\varepsilon$, 0, 00, 000, ...
Regular Expressions

**Examples:** In all examples $\Sigma = \{0, 1\}$.

- $0^*$ ... the language containing the only word $0$
- $01^*$ ... the language containing the only word $01$
- $0+1^*$ ... the language containing two words $0$ and $1$
- $0^*1^*$ ... the language containing words $\epsilon, 0, 00, 000, \ldots$
- $(01)^*$ ... the language containing words $\epsilon, 01, 0101, 010101, \ldots$
Regular Expressions

**Examples:** In all examples $\Sigma = \{0, 1\}$.

- $0^*$ ... the language containing the only word $0$
- $01^*$ ... the language containing the only word $01$
- $0 + 1^*$ ... the language containing two words $0$ and $1$
- $(01)^*$ ... the language containing words $\varepsilon, 01, 0101, 010101, \ldots$
- $(0 + 1)^*$ ... the language containing all words over the alphabet $\{0, 1\}$
Examples: In all examples \( \Sigma = \{0, 1\} \).

- \( 0 \) \ldots the language containing the only word 0
- \( 01 \) \ldots the language containing the only word 01
- \( 0 + 1 \) \ldots the language containing two words 0 and 1
- \( 0^* \) \ldots the language containing words \( \epsilon, 0, 00, 000, \ldots \)
- \( (01)^* \) \ldots the language containing words \( \epsilon, 01, 0101, 010101, \ldots \)
- \( (0 + 1)^* \) \ldots the language containing all words over the alphabet \{0, 1\}
- \( (0 + 1)^*00 \) \ldots the language containing all words ending with 00
Examples: In all examples $\Sigma = \{0, 1\}$.

- $0 \ldots$ the language containing the only word 0
- $01 \ldots$ the language containing the only word 01
- $0 + 1 \ldots$ the language containing two words 0 and 1
- $0^* \ldots$ the language containing words $\epsilon, 0, 00, 000, \ldots$
- $(01)^* \ldots$ the language containing words $\epsilon, 01, 0101, 010101, \ldots$
- $(0 + 1)^* \ldots$ the language containing all words over the alphabet $\{0, 1\}$
- $(0 + 1)^*00 \ldots$ the language containing all words ending with 00
- $(01)^*111(01)^* \ldots$ the language containing all words that contain a subword 111 preceded and followed by an arbitrary number of copies of the word 01
\((0 + 1)^*00 + (01)^*111(01)^*\) ... the language containing all words that either end with 00 or contain a subwords 111 preceded and followed with some arbitrary number of words 01
\[(0 + 1)^*00 + (01)^*111(01)^* \] ... the language containing all words that either end with 00 or contain a subwords 111 preceded and followed with some arbitrary number of words 01

\[(0 + 1)^*1(0 + 1)^* \] ... the language of all words that contain at least one occurrence of symbol 1
\((0 + 1)^*00 + (01)^*111(01)^*\) \ldots the language containing all words that either end with 00 or contain a subwords 111 preceded and followed with some arbitrary number of words 01

\((0 + 1)^*1(0 + 1)^*\) \ldots the language of all words that contain at least one occurrence of symbol 1

\(0^*(10^*10^*)^*\) \ldots the language containing all words with an even number of occurrences of symbol 1
Proposition

Every language that can be represented by a regular expression is regular (i.e., it is accepted by some finite automaton).

Proof: It is sufficient to show how to construct for a given regular expression $\alpha$ a finite automaton accepting the language \([\alpha]\).

The construction is recursive and proceeds by the structure of the expression $\alpha$:

- If $\alpha$ is a elementary expression (i.e., $\emptyset$, $\varepsilon$ or $a$):
  - We construct the corresponding automaton directly.

- If $\alpha$ is of the form $(\beta + \gamma)$, $(\beta \cdot \gamma)$ or $(\beta^*)$:
  - We construct automata accepting languages \([\beta]\) and \([\gamma]\) recursively.
  - Using these two automata, we construct the automaton accepting the language \([\alpha]\).
Transformation of a Regular Expression to a Finite Automaton

The automata for the elementary expressions:

\[\emptyset\]
\[\varepsilon\]
\[a\]
Transformation of a Regular Expression to a Finite Automaton

The automata for the elementary expressions:

- $\emptyset$
- $\epsilon$
- $a$

The construction for the union:
Transformation of a Regular Expression to a Finite Automaton

The automata for the elementary expressions:

\[ \emptyset \]

\[ \varepsilon \]

\[ a \]

The construction for the union:
The construction for the concatenation:
Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:
Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:

The construction for the iteration:
Transformation of a Regular Expression to a Finite Automaton

The construction for the concatenation:

The construction for the iteration:
Example: The construction of an automaton for expression \(((0 + 1) \cdot 1)^*\):
Example: The construction of an automaton for expression \(((0 + 1) \cdot 1)^*\):
Example: The construction of an automaton for expression $((0 + 1) \cdot 1)^*$:
Example: The construction of an automaton for expression \(((0 + 1) \cdot 1)^*\):
Example: The construction of an automaton for expression $\left((0 + 1 \cdot 1)^*\right)$:
Example: The construction of an automaton for expression $((0 + 1) \cdot 1)^*$:
Example: The construction of an automaton for expression $((0 + 1) \cdot 1)^*$:
Transformation of a Regular Expression to a Finite Automaton

If an expression $\alpha$ consists of $n$ symbols (not counting parenthesis) then the resulting automaton has:

- at most $2n$ states,
- at most $4n$ transitions.

**Remark:** By transforming the generalized nondeterministic automaton into a deterministic one, the number of states can grow exponentially, i.e., the resulting automaton can have up to $2^{2n} = 4^n$ states.
Proposition

Every regular language can be represented by some regular expression.

Proof: It is sufficient to show how to construct for a given finite automaton $A$ a regular expression $\alpha$ such that $[\alpha] = L(A)$.

- We modify $A$ in such a way that ensures it has exactly one initial and exactly one accepting state.
- Its states will be removed one by one.
- Its transitions will be labelled with regular expressions.
- The resulting automaton will have only two states – the initial and the accepting, and only one transition labelled with the resulting regular expression.
The main idea: If a state $q$ is removed, for every pair of remaining states $q_j, q_k$ we extend the label on a transition from $q_j$ to $q_k$ by a regular expression representing paths from $q_j$ to $q_k$ going through $q$.

After removing of the state $q$:

$$q_j \xrightarrow{\alpha + \beta \gamma^* \delta} q_k$$
Example:
Transformation of an Automaton to a Regular Expression

Example:
Example:
Example:

$$s \rightarrow a(b + aa)^* \rightarrow f$$

$$3 \rightarrow b + a(b + aa)^*ab \rightarrow 3$$

$$\epsilon + (a + ba)(b + aa)^* \rightarrow f$$

$$bb + (a + ba)(b + aa)^*ab$$
Transformation of an Automaton to a Regular Expression

Example:

\[ a(b + aa)^+ \]
\[ (b + a(b + aa)^*ab) \]
\[ (bb + (a + ba)(b + aa)^*ab)^* \]
\[ (\varepsilon + (a + ba)(b + aa)^*) \]
Theorem

A language is regular iff it can be represented by a regular expression.
Minimization of Automata
Minimization of DFA

Let us assume a determinitic finite automaton \( A = (Q, \Sigma, \delta, q_0, F) \).

**Definition**

States \( q, q' \in Q \) are called **equivalent**, written \( q \sim q' \), if for every \( w \in \Sigma^* \) we have \( q \xrightarrow{w} F \) iff \( q' \xrightarrow{w} F \).

**Remark:** The notation \( q \xrightarrow{w} F \) denotes that \( q \xrightarrow{w} q'' \) for some \( q'' \in F \).
Minimization of DFA

We will use $q \not\sim q'$ to denote $\neg(q \sim q')$.

Note that $q \not\sim q'$ iff there exists some word $w \in \Sigma^*$ such that:

- $q \xrightarrow{w} F$ and not $q' \xrightarrow{w} F$, or
- $q' \xrightarrow{w} F$ and not $q \xrightarrow{w} F$.

We will call such word $w$ distinguishing word for states $q$ and $q'$.

So $q \sim q'$ iff there is no distinguishing word for $q$ and $q'$.
Minimization of DFA

It is easy to see that $\sim$ is an equivalence relation on the set of states $Q$:

- It is *reflexive*: For every $q \in Q$ we have $q \sim q$.
- It is *symmetric*: $q \sim q'$ implies $q' \sim q$.
- It is *transitive*: If $q \sim q'$ and $q' \sim q''$ then $q \sim q''$.

We will see later how the relation $\sim$ can be computed efficiently.

We can note that replacing a transition $q \xrightarrow{a} q'$ with a transition $q \xrightarrow{a} q''$, where $q' \sim q''$, does not affect the language accepted by the automaton.

Similarly, if we replace the original initial state $q_0$ with some other initial state $q'_0$ such $q_0 \sim q'_0$, the language is not affected.

Basically, all states equivalent wrt $\sim$ can be “merged” into one state.
There are some other obvious properties of relation $\sim$:

For each $q_1, q_2 \in Q$ such that $q_1 \sim q_2$ we have:

- $q_1 \in F$ iff $q_2 \in F$
  - If $q_1 \in F$ and $q_2 \notin F$, or if $q_1 \notin F$ and $q_2 \in F$, then $\varepsilon$ is a distinguishing word for $q_1$ and $q_2$.

- It holds for each $a \in \Sigma$ and each $q_1', q_2' \in Q$ such that $q_1 \xrightarrow{a} q_1'$ and $q_2 \xrightarrow{a} q_2'$ that
  $$q_1' \sim q_2'.$$
  - If $q_1' \not\sim q_2'$ then there is some distinguishing word $w \in \Sigma^*$ for $q_1'$ and $q_2'$. But then $aw$ is be a distinguishing word for $q_1$ and $q_2$. 
This can be easily extended to arbitrary words:

- If \( q_1 \sim q_2 \) and \( q_1 \xrightarrow{w} q'_1 \) and \( q_2 \xrightarrow{w} q'_2 \) for \( w \in \Sigma^* \) then

  \[ q'_1 \sim q'_2. \]

- Otherwise there is some distinguishing word \( w' \in \Sigma^* \) for \( q'_1 \) and \( q'_2 \), which implies that \( ww' \) is a distinguishing word for \( q_1 \) and \( q_2 \).
Note also that the opposite claim is true:

For every \( q_1, q_2 \in Q \) such that:

- \( q_1 \in F \) iff \( q_2 \in F \), and
- for each \( a \in \Sigma \) and each \( q_1', q_2' \in Q \) such that \( q_1 \xrightarrow{a} q_1' \) and \( q_2 \xrightarrow{a} q_2' \) it is the case that \( q_1' \sim q_2' \),

it holds that \( q_1 \sim q_2 \).
Minimization of DFA

Z. Sawa (TU Ostrava)
Theoretical Computer Science
September 13, 2019 115 / 490
Minimization of DFA

\[ 4 \sim 8, \text{ so states 4 and 8 can be merged.} \]
4 \sim 8$, so states 4 and 8 can be merged.
Minimization of DFA

- $4 \sim 8$, so states 4 and 8 can be merged.
- $3 \sim 7$, so states 3 and 7 can be merged.
Minimization of DFA

- $4 \sim 8$, so states 4 and 8 can be merged.
- $3 \sim 7$, so states 3 and 7 can be merged.
Minimization of DFA

- $4 \sim 8$, so states 4 and 8 can be merged.
- $3 \sim 7$, so states 3 and 7 can be merged.
- $2 \sim 6$, so states 2 and 6 can be merged.
Minimization of DFA

- $4 \sim 8$, so states $4$ and $8$ can be merged.
- $3 \sim 7$, so states $3$ and $7$ can be merged.
- $2 \sim 6$, so states $2$ and $6$ can be merged.
Minimization of DFA

- $4 \sim 8$, so states 4 and 8 can be merged.
- $3 \sim 7$, so states 3 and 7 can be merged.
- $2 \sim 6$, so states 2 and 6 can be merged.
- $1 \sim 5$, so states 1 and 5 can be merged.
Minimization of DFA

- $4 \sim 8$, so states 4 and 8 can be merged.
- $3 \sim 7$, so states 3 and 7 can be merged.
- $2 \sim 6$, so states 2 and 6 can be merged.
- $1 \sim 5$, so states 1 and 5 can be merged.
Minimization of DFA

Since $\sim$ is an equivalence, there is a corresponding partition, i.e., a set $\{S_1, S_2, \ldots, S_k\}$ of subsets of $Q$ (i.e., $S_i \subseteq Q$ for each $i \in \{1, \ldots, k\}$) such that:

- $S_1 \cup S_2 \cup \cdots \cup S_k = Q$
- All sets $S_1, S_2, \ldots, S_k$ are pairwise disjoint, i.e., $S_i \cap S_j = \emptyset$ whenever $i \neq j$.

The sets $S_1, S_2, \ldots, S_k$ are called equivalence classes.

We will use $[q]$ to denote the equivalence class containing state $q \in Q$, i.e.,

$$[q] = \{ q' \in Q \mid q \sim q' \}.$$
Minimization of DFA

As follows from the previous discussion, the following holds for equivalence classes:

- For each $q \in Q$ we have $q \in F$ iff $q' \in F$ holds for every $q' \in [q]$.
- If $q_1 \xrightarrow{a} q'_1$ for some $a \in \Sigma$ and $q'_1 \in Q$, then for every $q_2 \in [q_1]$ there is some $q'_2 \in Q$ such that $q'_1 \sim q'_2$.

So if $q_1 \sim q_2$ then for each $a \in \Sigma$ we have $\delta(q_1, a) \sim \delta(q_2, a)$, or, in other words,

$$[q_1] = [q_2] \text{ implies } [\delta(q_1, a)] = [\delta(q_2, a)].$$
Minimization of DFA

By merging all equivalent states of an automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ we obtain so called quotient automaton $\mathcal{A}' = (Q', \Sigma, \delta', q'_0, F')$ where:

- $Q'$ is the set of equivalence classes of $Q$ wrt $\sim$, i.e., $Q' = \{[q] \mid q \in Q\}$.
- $\delta'$ is defined as follows: for $q \in Q$ and $a \in \Sigma$ is
  \[ \delta'([q], a) = [\delta(q, a)] . \]
- $q'_0 = [q_0]$
- $F' = \{[q] \mid q \in F\}$

It can be easily shown that $L(\mathcal{A}') = L(\mathcal{A})$. 
Minimization of DFA

Definition

A DFA $\mathcal{A}$ is called **reduced** if:

- It has no unreachable states.
- It holds for each $q, q' \in Q$ that $q \not\sim q'$ whenever $q \neq q'$
  (i.e., $q \sim q'$ implies $q = q'$).

For a given DFA $\mathcal{A}$ we can remove its unreachable states and construct a quotient automaton $\mathcal{A}'$ for the resulting automaton.

It is clear that $L(\mathcal{A}') = L(\mathcal{A})$ and $\mathcal{A}'$ is a reduced automaton.
Deterministic finite automata \( \mathcal{A} = (Q, \Sigma, \delta, q_0, F) \) and \( \mathcal{A}' = (Q', \Sigma, \delta', q'_0, F') \) are called **isomorphic** if there is an isomorphism between them, i.e., a function \( f : Q \to Q' \) such that:

- \( f \) is a bijection
- for each \( q \in Q \) and \( a \in \Sigma \) we have \( \delta'(f(q), a) = f(\delta(q, a)) \)
- \( q'_0 = f(q_0) \)
- for each \( q \in Q \) we have \( f(q) \in F' \) iff \( q \in F \)

**Remark:** Informally, \( \mathcal{A} \) and \( \mathcal{A}' \) are the same automaton, except the states of them are named differently.
Theorem
For each regular language \( L \) all reduced automata \( \mathcal{A} \), such that \( L(\mathcal{A}) = L \), are isomorphic.

Corollary
If \( \mathcal{A} \) is a reduced DFA and \( \mathcal{A} \) has \( n \) states then every DFA accepting the language \( L(\mathcal{A}) \) has at least \( n \) states.
Minimization of DFA

Now we will discuss the algorithm for computation of $\sim$ (resp. of the corresponding partition).

For this purpose it is useful to introduce auxiliary relations $\sim_0, \sim_1, \sim_2, \ldots$ on the set of states $Q$.

**Definition**

For $i \in \mathbb{N}$, the relation $\sim_i \subseteq Q \times Q$ is defined as follows:

$q \sim_i q'$ iff for each word $w \in \Sigma^*$ such that $|w| \leq i$ we have

$q \xrightarrow{w} F$ iff $q' \xrightarrow{w} F$.

**Remark:** It is easy to check that for each $i \in \mathbb{N}$, the relation $\sim_i$ is an equivalence.
Minimization of DFA

Some properties of relations \(\sim_0, \sim_1, \sim_2, \ldots\):

- For each \(i \in \mathbb{N}\) we have \(\sim_i \supseteq \sim_{i+1}\), i.e., for all \(q, q' \in Q\), \(q \sim_{i+1} q'\) implies \(q \sim_i q'\) (and so \(q \not\sim_i q'\) implies \(q \not\sim_{i+1} q'\)).

- \(q \sim_0 q'\) iff either \(q \in F\) and \(q' \in F\), or \(q \not\in F\) and \(q' \not\in F\).

- For each \(i \in \mathbb{N}\) and \(q_1, q_2 \in Q\) we have \(q_1 \sim_{i+1} q_2\) iff
  - \(q_1 \sim_i q_2\), and
  - for all \(a \in \Sigma\) we have \(\delta(q_1, a) \sim_i \delta(q_2, a)\).
Minimization of DFA

So we have a sequence of relations $\sim_0, \sim_1, \sim_2, \ldots$ on $Q$ where

$$\sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \cdots.$$ 

Note that if $\sim_i = \sim_{i+1}$ for some $i \in \mathbb{N}$ then $\sim_j = \sim_i$ for each $j \geq i$, i.e.,

$$\sim_i = \sim_{i+1} = \sim_{i+2} = \sim_{i+3} = \cdots.$$ 

Because $Q$ is a finite set, there must be such $i \in \mathbb{N}$ where $\sim_i = \sim_{i+1}$. 
Let $k \in \mathbb{N}$ be the smallest number such that $\sim_k = \sim_{k+1}$.

It follows that if some states $q$ and $q'$ cannot be distinguished by a word of length at most $k$ (i.e., $q \sim_k q'$), then they cannot be distinguished by a word of arbitrary length $i$ (i.e., $q \sim_i q'$ for each $i \in \mathbb{N}$).

But this means that they cannot be distinguished by any word, i.e., that $q \sim q'$.

**Lemma**

Let $k \in \mathbb{N}$ be the smallest number such that $\sim_k = \sim_{k+1}$. Then $\sim_k = \sim$, i.e., for each $q, q' \in Q$ we have

$$q \sim q' \iff q \sim_k q'.$$
Minimization of DFA

Z. Sawa (TU Ostrava)

Theoretical Computer Science
September 13, 2019 126 / 490
Minimization of DFA

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>←</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>→</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>←</td>
<td>8</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019  127 / 490
Minimization of DFA

Relation $\sim_0$:

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>←4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>→5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>←8</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>→5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>II</td>
<td>←4</td>
</tr>
<tr>
<td></td>
<td>←8</td>
</tr>
</tbody>
</table>
Minimization of DFA

Relation $\sim_0$:

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>$\leftarrow$ 4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>$\rightarrow$ 5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>$\leftarrow$ 8</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leftarrow$ 4</td>
<td>$\leftarrow$ 4</td>
<td>$\leftarrow$ 4</td>
</tr>
<tr>
<td>$\leftarrow$ 8</td>
<td>$\leftarrow$ 8</td>
<td>$\leftarrow$ 8</td>
</tr>
</tbody>
</table>
### Minimization of DFA

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**Relation \( \sim_0 \):**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>I</td>
</tr>
<tr>
<td>I</td>
<td>2</td>
<td>I</td>
</tr>
<tr>
<td>II</td>
<td>3</td>
<td>I</td>
</tr>
<tr>
<td>III</td>
<td>4</td>
<td>I</td>
</tr>
<tr>
<td>IV</td>
<td>8</td>
<td>I</td>
</tr>
</tbody>
</table>

**Relation \( \sim_1 \):**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>III</td>
<td>1</td>
</tr>
<tr>
<td>III</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>IV</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>II</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>II</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>II</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>II</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>II</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Minimization of DFA

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>←4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>→5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>←8</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

Relation $\sim_0$:

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1</td>
<td>I</td>
</tr>
<tr>
<td>2</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>3</td>
<td>II</td>
<td>I</td>
</tr>
<tr>
<td>→5</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>6</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>7</td>
<td>II</td>
<td>I</td>
</tr>
<tr>
<td>I I</td>
<td>←4</td>
<td>II</td>
</tr>
<tr>
<td>←8</td>
<td>II</td>
<td>I</td>
</tr>
</tbody>
</table>

Relation $\sim_1$:

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>III</td>
<td>1</td>
<td>III</td>
</tr>
<tr>
<td>2</td>
<td>III</td>
<td>IV</td>
</tr>
<tr>
<td>→5</td>
<td>III</td>
<td>III</td>
</tr>
<tr>
<td>6</td>
<td>III</td>
<td>IV</td>
</tr>
<tr>
<td>IV</td>
<td>3</td>
<td>II</td>
</tr>
<tr>
<td>7</td>
<td>II</td>
<td>IV</td>
</tr>
<tr>
<td>II</td>
<td>←4</td>
<td>II</td>
</tr>
<tr>
<td>←8</td>
<td>II</td>
<td>II</td>
</tr>
</tbody>
</table>
Minimization of DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>←4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>→5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>←8</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

Relation $\sim_1$:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>III</td>
<td>1</td>
<td>III</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>III</td>
</tr>
<tr>
<td></td>
<td>→5</td>
<td>III</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>III</td>
</tr>
<tr>
<td>IV</td>
<td>3</td>
<td>II</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>II</td>
</tr>
<tr>
<td>II</td>
<td>←4</td>
<td>II</td>
</tr>
<tr>
<td></td>
<td>←8</td>
<td>II</td>
</tr>
</tbody>
</table>
### Minimization of DFA

<table>
<thead>
<tr>
<th></th>
<th>(a)</th>
<th>(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

| Relation \(\sim_1\): |
|---|---|---|---|
| \(\text{III}\) | 1 | \(\text{III}\) | \(\text{III}\) |
| 2 | \(\text{III}\) | \(\text{IV}\) |
| \(\rightarrow 5\) | \(\text{III}\) | \(\text{III}\) |
| 6 | \(\text{III}\) | \(\text{IV}\) |
| \(\rightarrow 5\) | \(\text{IV}\) | \(\text{III}\) |
| \(\leftarrow 4\) | \(\text{II}\) | \(\text{IV}\) |
| \(\leftarrow 8\) | \(\text{II}\) | \(\text{II}\) |

| Relation \(\sim_2\): |
|---|---|---|
| \(\text{V}\) | 1 |   |
| \(\rightarrow 5\) |   |   |
| \(\text{VI}\) | 2 |   |
| 6 |   |   |
| \(\text{IV}\) | 3 |   |
| 7 |   |   |
| \(\leftarrow 4\) |   |   |
| \(\leftarrow 8\) |   |   |

Z. Sawa (TU Ostrava)
Minimization of DFA

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>a</th>
<th></th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>III</td>
<td>1</td>
<td>III</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>3</td>
<td>III</td>
<td>2</td>
<td>IV</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>III</td>
<td>5</td>
<td>III</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>4</td>
<td>IV</td>
<td>3</td>
<td>IV</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
<td>IV</td>
<td>7</td>
<td>IV</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>7</td>
<td>II</td>
<td>4</td>
<td>II</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>3</td>
<td>II</td>
<td>8</td>
<td>II</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>4</td>
<td>II</td>
<td></td>
<td>II</td>
</tr>
</tbody>
</table>

Relation $\sim_1$:

Relation $\sim_2$:
Minimization of DFA

Relation $\sim_2$:

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>←4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>→5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>←8</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>1</td>
<td>VI</td>
</tr>
<tr>
<td>VI</td>
<td>2</td>
<td>VI</td>
</tr>
<tr>
<td>VI</td>
<td>3</td>
<td>II</td>
</tr>
<tr>
<td>←4</td>
<td>4</td>
<td>II</td>
</tr>
<tr>
<td>←8</td>
<td>8</td>
<td>II</td>
</tr>
</tbody>
</table>
Minimization of DFA

Relation $\sim_2$:

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>←4</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>→5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>←8</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

\[
\begin{array}{ccc}
V & \rightarrow 5 & VI \\
VI & 2 & VI \\
IV & 3 & II \\
II & 4 & II \\
\end{array}
\]

\[
\begin{array}{ccc}
VI & \rightarrow 5 & V \\
VI & 6 & IV \\
II & 7 & IV \\
II & 8 & II \\
\end{array}
\]

\[
\begin{array}{ccc}
V & 1 & VI \\
VI & 2 & IV \\
II & 3 & II \\
II & 4 & II \\
\end{array}
\]

\[
\begin{array}{ccc}
VI & 1 & V \\
VI & 2 & IV \\
II & 3 & II \\
II & 4 & II \\
\end{array}
\]

\[
\begin{array}{ccc}
VI & 1 & V \\
II & 2 & II \\
II & 3 & II \\
II & 4 & II \\
\end{array}
\]
An algorithm for checking equivalence of deterministic finite automata \( A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1) \) and \( A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2) \):

- Construct an automaton \( A \) as the disjoint union of \( A_1 \) and \( A_2 \) where:
  - \( Q = Q_1 \cup Q_2 \) (without loss of generality we can assume \( Q_1 \cap Q_2 = \emptyset \))
  - For \( q \in Q \) and \( a \in \Sigma \)
    
    \[
    \delta(q, a) = \begin{cases} 
      \delta_1(q, a) & \text{if } q \in Q_1 \\
      \delta_2(q, a) & \text{if } q \in Q_2
    \end{cases}
    \]

- \( F = F_1 \cup F_2 \)
- (An initial state of \( A \) is not important.)

- Check if \( q_{01} \sim q_{02} \) in \( A \).
Nonregular Languages
Nonregular Languages

Not all languages are regular.
There are languages for which there exist no finite automata accepting them.

Examples of nonregular languages:

- $L_1 = \{a^n b^n \mid n \geq 0\}$
- $L_2 = \{ww \mid w \in \{a, b\}^*\}$
- $L_3 = \{ww^R \mid w \in \{a, b\}^*\}$

Remark: The existence of nonregular languages is already apparent from the fact that there are only countably many (nonisomorphic) automata working over some alphabet $\Sigma$ but there are uncountably many languages over the alphabet $\Sigma$. 
Nonregular Languages

How to prove that some language $L$ is not regular?

A language is not regular if there is no automaton (i.e., it is not possible to construct an automaton) accepting the language.

But how to prove that something does not exist?
Nonregular Languages

How to prove that some language $L$ is not regular?

A language is not regular if there is no automaton (i.e., it is not possible to construct an automaton) accepting the language.

But how to prove that something does not exist?

**The answer:** By contradiction.

E.g., we can assume there is some automaton $A$ accepting the language $L$, and show that this assumption leads to a contradiction.
We show that language $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

The proof by contradiction. Let us assume there exists a DFA $A = (Q, \Sigma, \delta, q_0, F)$ such that $L(A) = L$. 

Z. Sawa (TU Ostrava) Theoretical Computer Science September 13, 2019 132 / 490
We show that language $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

The proof by contradiction. Let us assume there exists a DFA $A = (Q, \Sigma, \delta, q_0, F)$ such that $L(A) = L$.

Let $|Q| = n$. 

\[ L = \{a^n b^n \mid n \geq 0\} \]
Nonregular Languages

We show that language $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

The proof by contradiction.
Let us assume there exists a DFA $A = (Q, \Sigma, \delta, q_0, F)$ such that $L(A) = L$.
Let $|Q| = n$.
Consider word $z = a^n b^n$. 
Nonregular Languages

We show that language $L = \{a^n b^n \mid n \geq 0\}$ is not regular.

The proof by contradiction.
Let us assume there exists a DFA $A = (Q, \Sigma, \delta, q_0, F)$ such that $L(A) = L$.

Let $|Q| = n$.

Consider word $z = a^n b^n$.

Since $z \in L$, there must be an accepting computation of the automaton $A$

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \cdots \xrightarrow{a} q_{n-1} \xrightarrow{a} q_n \xrightarrow{b} q_{n+1} \xrightarrow{b} \cdots \xrightarrow{b} q_{2n-1} \xrightarrow{b} q_{2n}$$

where $q_0$ is an initial state, and $q_{2n} \in F$. 
Consider now the first $n + 1$ states of the computation

$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \cdots \xrightarrow{a} q_{n-1} \xrightarrow{a} q_n \xrightarrow{b} q_{n+1} \xrightarrow{b} \cdots \xrightarrow{b} q_{2n-1} \xrightarrow{b} q_{2n}$

i.e., the sequence of states $q_0, q_1, \ldots, q_n$.

It is obvious that all states in this sequence can not be pairwise different, since $|Q| = n$ and the sequence has $n + 1$ elements.

This means that there exists a state $q \in Q$ which occurs (at least) twice in the sequence.
Nonregular Languages

Consider now the first $n + 1$ states of the computation

$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \cdots \xrightarrow{a} q_{n-1} \xrightarrow{a} q_n \xrightarrow{b} q_{n+1} \xrightarrow{b} \cdots \xrightarrow{b} q_{2n-1} \xrightarrow{b} q_{2n}$

i.e., the sequence of states $q_0, q_1, \ldots, q_n$.

It is obvious that all states in this sequence can not be pairwise different, since $|Q| = n$ and the sequence has $n + 1$ elements.

This means that there exists a state $q \in Q$ which occurs (at least) twice in the sequence.

It is an application of so called **pigeonhole principle**.

**Pigeonhole principle**

If we have $n + 1$ pigeons in $n$ holes then there is at least one hole containing at least two pigeons.
Consider now the first \( n + 1 \) states of the computation

\[
q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \cdots \xrightarrow{a} q_{n-1} \xrightarrow{a} q_n \xrightarrow{b} q_{n+1} \xrightarrow{b} \cdots \xrightarrow{b} q_{2n-1} \xrightarrow{b} q_{2n}
\]
i.e., the sequence of states \( q_0, q_1, \ldots, q_n \).

It is obvious that all states in this sequence can not be pairwise different, since \( |Q| = n \) and the sequence has \( n + 1 \) elements.

This means that there exists a state \( q \in Q \) which occurs (at least) twice in the sequence.

I.e., there are indexes \( i, j \) such that \( 0 \leq i < j \leq n \) and

\[
q_i = q_j
\]

which means that the automaton \( A \) must go through a cycle when reading the symbols \( a \) in the word \( z = a^n b^n \).
The word $z = a^n b^n$ can be divided into three parts $u, v, w$ such that $z = uvw$:

$$u = a^i \quad v = a^{j-i} \quad w = a^{n-j} b^n$$
For the words $u = a^i$, $v = a^{j-i}$, and $w = a^{n-j}b^n$ we have

\[ q_0 \xrightarrow{u} q_i \quad q_i \xrightarrow{v} q_j \quad q_j \xrightarrow{w} q_{2n} \]

Let $r$ be the length of the word $v$, i.e., $r = j - i$ (obviously $r > 0$, due to $i < j$).

Since $q_i = q_j$, the automaton accepts word $uw = a^{n-r}b^n$ that does not belong to $L$:

\[ q_0 \xrightarrow{u} q_i \xrightarrow{w} q_{2n} \]

The word $uvvw = a^{n+r}b^n$, that also does not belong to $L$, is accepted too:

\[ q_0 \xrightarrow{u} q_i \xrightarrow{v} q_i \xrightarrow{v} q_i \xrightarrow{w} q_{2n} \]
Similarly we can show that every word of the form $uvvvv \cdots vvw$, i.e., of the form $uv^k w$ for some $k \geq 0$, is accepted by the automaton $A$:

$$ q_0 \xrightarrow{u} q_i \xrightarrow{v} q_i \xrightarrow{v} q_i \xrightarrow{v} \cdots \xrightarrow{v} q_i \xrightarrow{v} q_i \xrightarrow{w} q_{2n} $$

A word of the form $uv^k w$ looks as follows: $a^{n-r+rk} b^n$.

Since $r > 0$, the following equivalence holds only for $k = 1$:

$$ n - r + rk = n $$

This means that if $k \geq 1$ then $uv^k w$ does not belong to the language $L$. However, the automaton $A$ accepts each such word, which is a contradiction with the assumption that $L(A) = \{a^n b^n \mid n \geq 0\}$. 
Pumping Lemma

Let us assume that language $L$ is accepted by some particular automaton $A$, i.e., $L = L(A)$.

Let us consider some arbitrary word $z \in L$ where $z = a_1a_2 \cdots a_k$.

Since automaton $A$ accepts word $z$, there must be some accepting computation of the automaton, i.e., a sequence of states:

$$q_0, q_1, q_2, \ldots, q_k$$

of length $k + 1$ where

- $q_0$ is an initial state
- $q_{i-1} \xrightarrow{a_i} q_i$ for each $i \in \{1, 2, \ldots, k\}$
- $q_k$ is an accepting state
Pumping Lemma

Let us assume that $A$ has $n$ states (i.e., $|Q| = n$), and that $|z| \geq n$. Since $|z| = k$, the computation of automaton $A$ over word $z$ forms a sequence, whose length is at least $n + 1$, that contains at most $n$ different states:

$$q_0, q_1, q_2, \ldots, q_{k-1}, q_k$$

It follows that there must be at least one state $q$ that occurs at least twice in this sequence (recall the pigeonhole principle).
Pumping Lemma

Let us say that the repeated state occurs on positions \( i \) and \( j \), i.e., \( q_i = q_j \) where \( i < j \).

\[ q_0, \cdots, q_i, \cdots, q_j, \cdots, q_k \]

**Remark:** It is obvious that in fact we can find \( i \) and \( j \) such that \( i < j \leq n \).

The word \( z \) can be divided into three parts:

\[
\begin{align*}
\underbrace{a_1 \cdots a_j}_{u} & \quad \underbrace{a_{i+1} \cdots a_j}_{v} & \quad \underbrace{a_{j+1} \cdots a_k}_{w}
\end{align*}
\]

- \( q_0 \xrightarrow{u} q_i \)
- \( q_i \xrightarrow{v} q_j \) (and so also \( q_i \xrightarrow{v} q_i \) since \( q_j = q_i \))
- \( q_j \xrightarrow{w} q_k \) (and so also \( q_i \xrightarrow{w} q_k \) since \( q_j = q_i \))
Pumping Lemma

Consider now words:

\[ a_1 \cdots a_j \quad a_{j+1} \cdots a_k \]

\[ u \quad w \]

\[ a_1 \cdots a_i \quad a_{i+1} \cdots a_j \quad a_{i+1} \cdots a_j \quad a_{j+1} \cdots a_k \]

\[ u \quad v \quad v \quad w \]

\[ a_1 \cdots a_i \quad a_{i+1} \cdots a_j \quad a_{i+1} \cdots a_j \quad a_{i+1} \cdots a_j \quad a_{j+1} \cdots a_k \]

\[ u \quad v \quad v \quad v \quad w \]

\[ \ldots \]

It is obvious that \( A \) accepts all of them because

- \( q_0 \xrightarrow{u} q_i \)
- \( q_i \xrightarrow{v} q_i \)
- \( q_i \xrightarrow{w} q_k \) where \( q_k \in F \)
Pumping Lemma

If language $L$ is regular then there exists $n \in \mathbb{N}$ such that every word $z \in L$ such that $|z| \geq n$ can be divided into subwords $u, v, w$ such that $z = uvw$, $|uv| \leq n$, $|v| \geq 1$, and for every $i \geq 0$ it holds that $uv^i w \in L$.

Formally:

If $L$ is regular then

$(\exists n \in \mathbb{N})(\forall z \in L \text{ s.t. } |z| \geq n)(\exists u, v, w \text{ s.t. } z = uvw, |uv| \leq n, |v| \geq 1)$

$(\forall i \geq 0) : uv^i w \in L$
We can take the contrapositive of the pumping lemma. \((A \Rightarrow B)\) is equivalent to \(\neg B \Rightarrow \neg A\).

If
\[
(\forall n \in \mathbb{N})(\exists z \in L \text{ s.t. } |z| \geq n)(\forall u, v, w \text{ s.t. } z = uvw, |uv| \leq n, |v| \geq 1)
(\exists i \geq 0) : uv^i w \notin L,
\]
then \(L\) is not regular.

So if we want to show that a language \(L\) is not regular, it is sufficient to show that \(L\) satisfies this condition.
Example: Let us consider language $L = \{a^i b^i \mid i \geq 0\}$.

- Let us assume that $L$ is accepted by some automaton with $n$ states.
- Let us consider word $z = a^n b^n$.
- Let us consider all possibilities how $z$ can be divided into three subwords $u, v, w$ satisfying conditions $|uv| \leq n$ and $|v| \geq 1$.

It is obvious that words $u$ and $v$ contain only symbols $a$. For every particular division there are some $j$ and $k$ such that $j + k \leq n$, $k \geq 1$, and

- $u = a^j$
- $v = a^k$
- $w = a^{n-(j+k)} b^n$

If we choose $i = 0$, we obtain $uv^i w = uw = a^{n-k} b^n$. Since $n - k < n$, we have $uv^i w \notin L$. 

Z. Sawa (TU Ostrava)
Remark: Proving that some first order logic formula with alternating universal and existential quantifiers can be viewed as game played by two players, Player A and Player B.

Player A chooses values of variables bound by existential quantifiers and Player B values of variables bound by universal quantifiers.

If we want to refute the given claim, it is sufficient to find a winning strategy for Player B.
Pumping Lemma

If $L$ is regular then

$$(\exists n \in \mathbb{N})(\forall z \in L \text{ s.t. } |z| \geq n)(\exists u, v, w \text{ s.t. } z = uvw, |uv| \leq n, |v| \geq 1)$$

$$((\forall i \geq 0) : uv^i w \in L).$$

The game for Pumping Lemma looks as follows:

1. Player A chooses some $n \in \mathbb{N}$.
2. Player B chooses a word $z$ such that $z \in L$ and $|z| \geq n$.
3. Player A chooses words $u, v, w$ such that $z = uvw, |uv| \leq n, |v| \geq 1$.
4. Player B chooses $i \geq 0$.
5. If $uv^i w \in L$ then Player A wins. If $uv^i w \notin L$ then Player B wins.

If Player B has a winning strategy in this game then $L$ is not regular.
Example: \( L = \{a^i b^i \mid i \geq 0\} \)

1. Player A chooses \( n > 0 \).
2. Player B chooses \( z = a^n b^n \).
3. Player A chooses words \( u, v, w \) such that \( z = uvw \), \(|uv| \leq n\), \(|v| \geq 1\).
4. Player B chooses \( i = 0 \).
5. Player B wins, since no matter what Player A does, we always have \( uv^i w \not\in L \) because a non-empty word \( z \) occurs in the part of word \( z \) consisting only of symbols \( a \), and when we omit it, we obtain a word of the form \( a^k b^n \) where \( k < n \), which does not belong to \( L \).
Context-Free Grammars
Example: We would like to describe a language of arithmetic expressions, containing expressions such as:

\[ 175 \quad (9+15) \quad (((10-4)*((1+34)+2))/(3+(-37))) \]

For simplicity we assume that:
- Expressions are fully parenthesized.
- The only arithmetic operations are "+", "-", "*", "/" and unary "-".
- Values of operands are natural numbers written in decimal — a number is represented as a non-empty sequence of digits.

Alphabet: \( \Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, (, )\} \)
Example (cont.): A description by an inductive definition:

- **Digit** is any of characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

- **Number** is a non-empty sequence of digits, i.e.:
  - If \( \alpha \) is a digit then \( \alpha \) is a number.
  - If \( \alpha \) is a digit and \( \beta \) is a number then also \( \alpha \beta \) is a number.

- **Expression** is a sequence of symbols constructed according to the following rules:
  - If \( \alpha \) is a number then \( \alpha \) is an expression.
  - If \( \alpha \) is an expression then also \( (-\alpha) \) is an expression.
  - If \( \alpha \) and \( \beta \) are expressions then also \( (\alpha+\beta) \) is an expression.
  - If \( \alpha \) and \( \beta \) are expressions then also \( (\alpha-\beta) \) is an expression.
  - If \( \alpha \) and \( \beta \) are expressions then also \( (\alpha*\beta) \) is an expression.
  - If \( \alpha \) and \( \beta \) are expressions then also \( (\alpha/\beta) \) is an expression.
Example (cont.): The same information that was described by the previous inductive definition can be represented by a context-free grammar:

New auxiliary symbols, called nonterminals, are introduced:

- **D** — stands for an arbitrary digit
- **C** — stands for an arbitrary number
- **E** — stands for an arbitrary expression

\[
\begin{align*}
D & \rightarrow 0 \\
D & \rightarrow 1 \\
D & \rightarrow 2 \\
D & \rightarrow 3 \\
D & \rightarrow 4 \\
D & \rightarrow 5 \\
D & \rightarrow 6 \\
D & \rightarrow 7 \\
D & \rightarrow 8 \\
D & \rightarrow 9 \\
C & \rightarrow D \\
E & \rightarrow C \\
E & \rightarrow (-E) \\
E & \rightarrow (E+E) \\
E & \rightarrow (E-E) \\
E & \rightarrow (E*E) \\
E & \rightarrow (E/E)
\end{align*}
\]
Example (cont.): Written in a more succinct way:

\[
\begin{align*}
D & \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
C & \rightarrow D \mid DC \\
E & \rightarrow C \mid (-E) \mid (E+E) \mid (E-E) \mid (E*E) \mid (E/E)
\end{align*}
\]
Example: A language where words are (possibly empty) sequences of expressions described in the previous example, where individual expressions are separated by commas (the alphabet must be extended with symbol “,”):

\[
\begin{align*}
S & \rightarrow T \mid \epsilon \\
T & \rightarrow E \mid E, T \\
D & \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
C & \rightarrow D \mid DC \\
E & \rightarrow C \mid (-E) \mid (E+E) \mid (E-E) \mid (E\times E) \mid (E/E)
\end{align*}
\]
Context-Free Grammars

Example: Statements of some programming language (a fragment of a grammar):

\[
S \rightarrow E; \mid T \mid \text{if} \ (E) \ S \mid \text{if} \ (E) \ S \ \text{else} \ S \\
\quad \mid \text{while} \ (E) \ S \mid \text{do} \ S \ \text{while} \ (E); \mid \text{for} \ (F; F; F) \ S \\
\quad \mid \text{return} \ F;
\]

\[
T \rightarrow \{ \ U \ \}
\]

\[
U \rightarrow \epsilon \mid SU
\]

\[
F \rightarrow \epsilon \mid E
\]

\[
E \rightarrow \ldots
\]

Remark:

- **S** — statement
- **T** — block of statements
- **U** — sequence of statements
- **E** — expression
- **F** — optional expression that can be omitted
Formally, a **context-free grammar** is a tuple

\[ G = (\Pi, \Sigma, S, P) \]

where:

- \( \Pi \) is a finite set of **nonterminal symbols** (nonterminals)
- \( \Sigma \) is a finite set of **terminal symbols** (terminals), where \( \Pi \cap \Sigma = \emptyset \)
- \( S \in \Pi \) is an **initial nonterminal**
- \( P \subseteq \Pi \times (\Pi \cup \Sigma)^* \) is a finite set of **rewrite rules**
Remarks:

- We will use uppercase letters $A, B, C, \ldots$ to denote nonterminal symbols.
- We will use lowercase letters $a, b, c, \ldots$ or digits $0, 1, 2, \ldots$ to denote terminal symbols.
- We will use lowercase Greek letters $\alpha, \beta, \gamma, \ldots$ to denote strings from $(\Pi \cup \Sigma)^*$.
- We will use the following notation for rules instead of $(A, \alpha)$

$$A \rightarrow \alpha$$

$A$ – left-hand side of the rule
$\alpha$ – right-hand side of the rule
**Example:** Grammar $G = (\Pi, \Sigma, S, P)$ where

- $\Pi = \{A, B, C\}$
- $\Sigma = \{a, b\}$
- $S = A$
- $P$ contains rules

\[
\begin{align*}
A & \rightarrow aBBb \\
A & \rightarrow AaA \\
B & \rightarrow \varepsilon \\
B & \rightarrow bCA \\
C & \rightarrow AB \\
C & \rightarrow a \\
C & \rightarrow b
\end{align*}
\]
Remark: If we have more rules with the same left-hand side, as for example

\[
A \rightarrow \alpha_1 \quad A \rightarrow \alpha_2 \quad A \rightarrow \alpha_3
\]

we can write them in a more succinct way as

\[
A \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3
\]

For example, the rules of the grammar from the previous slide can be written as

\[
A \rightarrow aBBb \mid AaA \\
B \rightarrow \varepsilon \mid bCA \\
C \rightarrow AB \mid a \mid b
\]
Context-Free Grammars

Grammars are used for generating words.

Example: \( G = (\Pi, \Sigma, A, P) \) where \( \Pi = \{A, B, C\} \), \( \Sigma = \{a, b\} \), and \( P \) contains rules

\[
A \rightarrow aBBb \mid AaA \\
B \rightarrow \varepsilon \mid bCA \\
C \rightarrow AB \mid a \mid b
\]

For example, the word \( abbabb \) can be in grammar \( G \) generated as follows:
Context-Free Grammars

Grammars are used for generating words.

**Example:** \( G = (\Pi, \Sigma, A, P) \) where \( \Pi = \{A, B, C\} \), \( \Sigma = \{a, b\} \), and \( P \) contains rules:

\[
A \to aBBb \mid AaA \\
B \to \epsilon \mid bCA \\
C \to AB \mid a \mid b
\]

For example, the word \( abbabb \) can be in grammar \( G \) generated as follows:

\( A \)
Context-Free Grammars

Grammars are used for generating words.

**Example:** $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and $P$ contains rules

- $A \rightarrow aBBb \mid AaA$
- $B \rightarrow \varepsilon \mid bCA$
- $C \rightarrow AB \mid a \mid b$

For example, the word $abbabb$ can be in grammar $G$ generated as follows:
Context-Free Grammars

Grammars are used for generating words.

Example: $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and $P$ contains rules

$A \rightarrow aBBb \mid AaA$

$B \rightarrow \varepsilon \mid bCA$

$C \rightarrow AB \mid a \mid b$

For example, the word $abbabb$ can be in grammar $G$ generated as follows:

$A \Rightarrow aBBb$
Context-Free Grammars

Grammars are used for generating words.

**Example:** \( G = (\Pi, \Sigma, A, P) \) where \( \Pi = \{A, B, C\} \), \( \Sigma = \{a, b\} \), and \( P \) contains rules

\[
A \rightarrow aBBb \mid AaA \\
B \rightarrow \varepsilon \mid bCA \\
C \rightarrow AB \mid a \mid b
\]

For example, the word \( abbabb \) can be in grammar \( G \) generated as follows:

\( A \Rightarrow aBBb \)
Grammars are used for generating words.

**Example:** $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and $P$ contains rules

\[
A \rightarrow aBBb \mid AaA \\
B \rightarrow \varepsilon \mid bCA \\
C \rightarrow AB \mid a \mid b
\]

For example, the word $abbabb$ can be in grammar $G$ generated as follows:

$A \Rightarrow aBBb$
Context-Free Grammars

Grammars are used for generating words.

**Example:** \( G = (\Pi, \Sigma, A, P) \) where \( \Pi = \{A, B, C\} \), \( \Sigma = \{a, b\} \), and \( P \) contains rules

\[
\begin{align*}
A & \rightarrow aBBb | AaA \\
B & \rightarrow \varepsilon | bCA \\
C & \rightarrow AB | a | b
\end{align*}
\]

For example, the word \( abbabb \) can be in grammar \( G \) generated as follows:

\[
A \Rightarrow aBBb \Rightarrow abCAAbb
\]
Context-Free Grammars

Grammars are used for generating words.

**Example:** $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and $P$ contains rules

\[
A \to aBBb \mid AaA \\
B \to \varepsilon \mid bCA \\
C \to AB \mid a \mid b
\]

For example, the word $abbabb$ can be in grammar $G$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb$
Grammars are used for generating words.

**Example:** $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and $P$ contains rules

$$
A \rightarrow aBBb \mid AaA \\
B \rightarrow \varepsilon \mid bCA \\
C \rightarrow AB \mid a \mid b
$$

For example, the word $abbabb$ can be in grammar $G$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCAABb$
Context-Free Grammars

Grammars are used for generating words.

**Example:** $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and $P$ contains rules

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

For example, the word $abbabb$ can be in grammar $G$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb$
Context-Free Grammars

Grammars are used for generating words.

**Example:** \( G = (\Pi, \Sigma, A, P) \) where \( \Pi = \{ A, B, C \} \), \( \Sigma = \{ a, b \} \), and \( P \) contains rules

\[
A \rightarrow aBBb \mid AaA \\
B \rightarrow \varepsilon \mid bCA \\
C \rightarrow AB \mid a \mid b
\]

For example, the word \( abbabb \) can be in grammar \( G \) generated as follows:

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBbBb
\]
Context-Free Grammars

Grammars are used for generating words.

**Example:** \( G = (\Pi, \Sigma, A, P) \) where \( \Pi = \{A, B, C\} \), \( \Sigma = \{a, b\} \), and \( P \) contains rules

\[
\begin{align*}
A & \rightarrow aBBb \mid AaA \\
B & \rightarrow \varepsilon \mid bCA \\
C & \rightarrow AB \mid a \mid b
\end{align*}
\]

For example, the word \( abbabb \) can be in grammar \( G \) generated as follows:
\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBbBb
\]
Context-Free Grammars

Grammars are used for generating words.

Example: \( G = (\Pi, \Sigma, A, P) \) where \( \Pi = \{A, B, C\} \), \( \Sigma = \{a, b\} \), and \( P \) contains rules

\[
\begin{align*}
A & \rightarrow aBBb \mid AaA \\
B & \rightarrow \epsilon \mid bCA \\
C & \rightarrow AB \mid a \mid b
\end{align*}
\]

For example, the word \( abbabb \) can be in grammar \( G \) generated as follows:

\[
\begin{align*}
A \Rightarrow aBBb & \Rightarrow abCABb \\
& \Rightarrow abCaBbBb \\
& \Rightarrow abCaBbBb
\end{align*}
\]
Context-Free Grammars

Grammars are used for generating words.

**Example:** $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and $P$ contains rules

\[
\begin{align*}
A & \rightarrow aBBb \mid AaA \\
B & \rightarrow \varepsilon \mid bCA \\
C & \rightarrow AB \mid a \mid b
\end{align*}
\]

For example, the word $abbabb$ can be in grammar $G$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb$
Context-Free Grammars

Grammars are used for generating words.

Example: \( G = (\Pi, \Sigma, A, P) \) where \( \Pi = \{A, B, C\} \), \( \Sigma = \{a, b\} \), and \( P \) contains rules

\[
A \rightarrow aBBb \mid AaA \\
B \rightarrow \varepsilon \mid bCA \\
C \rightarrow AB \mid a \mid b
\]

For example, the word \textit{abbabb} can be in grammar \( G \) generated as follows:

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abC aBbBb
\]
Context-Free Grammars

Grammars are used for generating words.

**Example:** \( G = (\Pi, \Sigma, A, P) \) where \( \Pi = \{A, B, C\} \), \( \Sigma = \{a, b\} \), and \( P \) contains rules

\[
\begin{align*}
A & \rightarrow aBBb \mid AaA \\
B & \rightarrow \epsilon \mid bCA \\
C & \rightarrow AB \mid a \mid b
\end{align*}
\]

For example, the word \( aabbrb \) can be in grammar \( G \) generated as follows:

\[
\begin{align*}
A & \Rightarrow aBBb \\
& \Rightarrow abCABb \\
& \Rightarrow abCaBBbBb \\
& \Rightarrow abCaBbBb \\
& \Rightarrow abbaBbBb
\end{align*}
\]
Context-Free Grammars

Grammars are used for generating words.

**Example:** \( G = (\Pi, \Sigma, A, P) \) where \( \Pi = \{A, B, C\} \), \( \Sigma = \{a, b\} \), and \( P \) contains rules

\[
\begin{align*}
A & \rightarrow aBBb \mid AaA \\
B & \rightarrow \varepsilon \mid bCA \\
C & \rightarrow AB \mid a \mid b
\end{align*}
\]

For example, the word \textit{abbabb} can be in grammar \( G \) generated as follows:

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb
\]
Context-Free Grammars

Grammars are used for generating words.

**Example:**  $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and $P$ contains rules

\[
\begin{align*}
A & \to aBBb \mid AaA \\
B & \to \epsilon \mid bCA \\
C & \to AB \mid a \mid b
\end{align*}
\]

For example, the word $abbabb$ can be in grammar $G$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb$
Context-Free Grammars

Grammars are used for generating words.

**Example:** \( G = (\Pi, \Sigma, A, P) \) where \( \Pi = \{ A, B, C \} \), \( \Sigma = \{ a, b \} \), and \( P \) contains rules

\[
\begin{align*}
A & \rightarrow aBBb \mid AaA \\
B & \rightarrow \varepsilon \mid bCA \\
C & \rightarrow AB \mid a \mid b
\end{align*}
\]

For example, the word \( abbabb \) can be in grammar \( G \) generated as follows:

\[
\begin{align*}
A \Rightarrow & \ aBBb \Rightarrow \ abCABb \Rightarrow \ abCaBBbBb \Rightarrow \ abCaBbBb \Rightarrow \ abbaBbBb \Rightarrow \\
& \ abbaBbb
\end{align*}
\]
Context-Free Grammars

Grammars are used for generating words.

**Example:** \(G = (\Pi, \Sigma, A, P)\) where \(\Pi = \{A, B, C\}\), \(\Sigma = \{a, b\}\), and \(P\) contains rules

\[
A \rightarrow aBBb \mid AaA \\
B \rightarrow \varepsilon \mid bCA \\
C \rightarrow AB \mid a \mid b
\]

For example, the word \(abbabb\) can be in grammar \(G\) generated as follows:

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb
\]
Context-Free Grammars

Grammars are used for generating words.

Example: \( G = (\Pi, \Sigma, A, P) \) where \( \Pi = \{A, B, C\} \), \( \Sigma = \{a, b\} \), and \( P \) contains rules

\[
A \rightarrow aBBb \mid AaA \\
B \rightarrow \varepsilon \mid bCA \\
C \rightarrow AB \mid a \mid b
\]

For example, the word \( abbabb \) can be in grammar \( G \) generated as follows:

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb
\]
Context-Free Grammars

Grammars are used for generating words.

Example:  \( G = (\Pi, \Sigma, A, P) \) where \( \Pi = \{A, B, C\} \), \( \Sigma = \{a, b\} \), and \( P \) contains rules

\[
\begin{align*}
A & \rightarrow aBBb \mid AaA \\
B & \rightarrow \varepsilon \mid bCA \\
C & \rightarrow AB \mid a \mid b
\end{align*}
\]

For example, the word \( abbabb \) can be in grammar \( G \) generated as follows:

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaB_bb \Rightarrow abbabb
\]
Context-Free Grammars

Grammars are used for generating words.

**Example:** $G = (\Pi, \Sigma, A, P)$ where $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$, and $P$ contains rules

\[
\begin{align*}
A & \rightarrow aBBb \mid AaA \\
B & \rightarrow \varepsilon \mid bCA \\
C & \rightarrow AB \mid a \mid b
\end{align*}
\]

For example, the word $abbabb$ can be in grammar $G$ generated as follows:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb \Rightarrow abbabb$
Context-Free Grammars

On strings from \((\Pi \cup \Sigma)^*\) we define relation \(\Rightarrow \subseteq (\Pi \cup \Sigma)^* \times (\Pi \cup \Sigma)^*\) such that

\[ \alpha \Rightarrow \alpha' \]

iff \(\alpha = \beta_1 A \beta_2\) and \(\alpha' = \beta_1 \gamma \beta_2\) for some \(\beta_1, \beta_2, \gamma \in (\Pi \cup \Sigma)^*\) and \(A \in \Pi\) where \((A \rightarrow \gamma) \in P\).

**Example:** If \((B \rightarrow bCA) \in P\) then

\[ aCBbA \Rightarrow aC bCA bA \]

**Remark:** Informally, \(\alpha \Rightarrow \alpha'\) means that it is possible to derive \(\alpha'\) from \(\alpha\) by one step where an occurrence of some nonterminal \(A\) in \(\alpha\) is replaced with the right-hand side of some rule \(A \rightarrow \gamma\) with \(A\) on the left-hand side.
Context-Free Grammars

On strings from \((\Pi \cup \Sigma)^*\) we define relation \(\Rightarrow \subseteq (\Pi \cup \Sigma)^* \times (\Pi \cup \Sigma)^*\) such that

\[ \alpha \Rightarrow \alpha' \]

iff \(\alpha = \beta_1 A \beta_2\) and \(\alpha' = \beta_1 \gamma \beta_2\) for some \(\beta_1, \beta_2, \gamma \in (\Pi \cup \Sigma)^*\) and \(A \in \Pi\) where \((A \rightarrow \gamma) \in P\).

Example: If \((B \rightarrow bCA) \in P\) then

\[ aC BbA \Rightarrow aC bC A bA \]

Remark: Informally, \(\alpha \Rightarrow \alpha'\) means that it is possible to derive \(\alpha'\) from \(\alpha\) by one step where an occurrence of some nonterminal \(A\) in \(\alpha\) is replaced with the right-hand side of some rule \(A \rightarrow \gamma\) with \(A\) on the left-hand side.
A **derivation** of length $n$ is a sequence $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$, where $\beta_i \in (\Pi \cup \Sigma)^*$, and where $\beta_{i-1} \Rightarrow \beta_i$ for all $1 \leq i \leq n$, which can be written more succinctly as

$$\beta_0 \Rightarrow \beta_1 \Rightarrow \beta_2 \Rightarrow \ldots \Rightarrow \beta_{n-1} \Rightarrow \beta_n$$

The fact that for given $\alpha, \alpha' \in (\Pi \cup \Sigma)^*$ and $n \in \mathbb{N}$ there exists some derivation $\beta_0 \Rightarrow \beta_1 \Rightarrow \beta_2 \Rightarrow \ldots \Rightarrow \beta_{n-1} \Rightarrow \beta_n$, where $\alpha = \beta_0$ and $\alpha' = \beta_n$, is denoted

$$\alpha \Rightarrow^n \alpha'$$

The fact that $\alpha \Rightarrow^n \alpha'$ for some $n \geq 0$, is denoted

$$\alpha \Rightarrow^* \alpha'$$

**Remark:** Relation $\Rightarrow^*$ is the reflexive and transitive closure of relation $\Rightarrow$ (i.e., the smallest reflexive and transitive relation containing relation $\Rightarrow$).
Sentential forms are those $\alpha \in (\Pi \cup \Sigma)^*$, for which

$$S \Rightarrow^* \alpha$$

where $S$ is the initial nonterminal.
A language $L(G)$ generated by a grammar $G = (\Pi, \Sigma, S, P)$ is the set of all words over alphabet $\Sigma$ that can be derived by some derivation from the initial nonterminal $S$ using rules from $P$, i.e.,

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$
Example: We want to construct a grammar generating the language

\[ L = \{a^n b^n \mid n \geq 0\} \]
Example: We want to construct a grammar generating the language

\[ L = \{ a^n b^n \mid n \geq 0 \} \]

Grammar \( G = (\Pi, \Sigma, S, P) \) where \( \Pi = \{ S \}, \Sigma = \{ a, b \}, \) and \( P \) contains

\[ S \rightarrow aSb \mid \varepsilon \]
Example: We want to construct a grammar generating the language

\[ L = \{a^n b^n \mid n \geq 0\} \]

Grammar \( G = (\Pi, \Sigma, S, P) \) where \( \Pi = \{S\} \), \( \Sigma = \{a, b\} \), and \( P \) contains

\[ S \rightarrow aSb \mid \varepsilon \]

\[
\begin{align*}
S & \Rightarrow \varepsilon \\
S & \Rightarrow aSb \Rightarrow ab \\
S & \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb \\
S & \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb \\
S & \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaaSbbbb \Rightarrow aaaaabbbb \\
& \quad \ldots
\end{align*}
\]
Example: We want to construct a grammar generating the language consisting of all palindroms over the alphabet \( \{a, b\} \), i.e.,

\[
L = \{ w \in \{a, b\}^* | w = w^R \}
\]

Remark: \( w^R \) denotes the reverse of a word \( w \), i.e., the word \( w \) written backwards.
Example: We want to construct a grammar generating the language consisting of all palindroms over the alphabet \( \{a, b\} \), i.e.,

\[
L = \{ w \in \{a, b\}^* \mid w = w^R \}
\]

Remark: \( w^R \) denotes the reverse of a word \( w \), i.e., the word \( w \) written backwards.

Solution:

\[
S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon
\]
**Example:** We want to construct a grammar generating the language consisting of all palindromes over the alphabet \( \{a, b\} \), i.e.,

\[
L = \{w \in \{a, b\}^* \mid w = w^R\}
\]

**Remark:** \( w^R \) denotes the reverse of a word \( w \), i.e., the word \( w \) written backwards.

**Solution:**

\[
S \rightarrow aSa \mid bSb \mid a \mid b \mid \epsilon
\]

\[
S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow abaaaba
\]
**Example:** We want to construct a grammar generating the language $L$ consisting of all correctly parenthesised sequences of symbols ‘(’ and ‘)’.

For example $(())(())(()) \in L$ but $)() \not\in L$. 
**Example:** We want to construct a grammar generating the language $L$ consisting of all correctly parenthesised sequences of symbols ‘(’ and ‘)’. For example $(())() ()() ()$ $\in L$ but $)()$ $\not\in L$.

**Solution:**

$$S \rightarrow \varepsilon \mid (S) \mid SS$$
**Example:** We want to construct a grammar generating the language $L$ consisting of all correctly parenthesised sequences of symbols ‘(’ and ‘)’.

For example $(())(()) \in L$ but $)() \notin L$.

**Solution:**

\[ S \rightarrow \varepsilon \mid (S) \mid SS \]

\[ S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow (SS)(S) \Rightarrow ((S)S)(S) \Rightarrow \\
((()S)(S) \Rightarrow (((S))(S))(S) \Rightarrow (((())(S))(S) \Rightarrow (((()))(S))(S) \Rightarrow \\
(((())))(SS) \Rightarrow \\
(((())))(()) \]

Z. Sawa (TU Ostrava)

Theoretical Computer Science

September 13, 2019 165 / 490
Example: We want to construct a grammar generating the language $L$ consisting of all correctly constructed arithmetic expressions where operands are always of the form ‘$a$’ and where symbols $+$ and $*$ can be used as operators.

For example $(a + a) * a + (a * a) \in L$. 
**Example:** We want to construct a grammar generating the language $L$ consisting of all correctly constructed arithmetic expressions where operands are always of the form ‘$a$’ and where symbols $+$ and $*$ can be used as operators.

For example $(a + a) * a + (a * a) \in L$.

**Solution:**

$$E \rightarrow a \mid E + E \mid E * E \mid (E)$$
**Example:** We want to construct a grammar generating the language $L$ consisting of all correctly constructed arithmetic expressions where operands are always of the form ‘$a$’ and where symbols $+$ and $*$ can be used as operators.

For example $(a + a) * a + (a * a) \in L$.

**Solution:**

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

$$E \Rightarrow E + E \Rightarrow E * E + E \Rightarrow (E) * E + E \Rightarrow (E + E) * E + E \Rightarrow (a + E) * E + E \Rightarrow (a + a) * E + E \Rightarrow (a + a) * a + E \Rightarrow (a + a) * a + (E) \Rightarrow (a + a) * a + (E * E) \Rightarrow (a + a) * a + (a * E) \Rightarrow (a + a) * a + (a * a)$$
\[ A \rightarrow aBBb \mid AaA \\
B \rightarrow \epsilon \mid bCA \\
C \rightarrow AB \mid a \mid b \]
A → aBBb | AaA
B → ε | bCA
C → AB | a | b
Derivation Tree

\[
\begin{align*}
A & \rightarrow aBBb \mid AaA \\
B & \rightarrow \epsilon \mid bCA \\
C & \rightarrow AB \mid a \mid b
\end{align*}
\]
$A \rightarrow aBBb \mid AaA$

$B \rightarrow \varepsilon \mid bCA$

$C \rightarrow AB \mid a \mid b$

$A \Rightarrow aBBb$
Derivation Tree

\[ A \rightarrow aBBb \mid AaA \]
\[ B \rightarrow \varepsilon \mid bCA \]
\[ C \rightarrow AB \mid a \mid b \]

\[ A \Rightarrow aBBb \]
\[ A \rightarrow aBBb \mid AaA \\
B \rightarrow \varepsilon \mid bCA \\
C \rightarrow AB \mid a \mid b \]

\[ A \Rightarrow aBbBb \]
Derivation Tree

\[
\begin{align*}
A & \rightarrow aBBb \mid AaA \\
B & \rightarrow \epsilon \mid bCA \\
C & \rightarrow AB \mid a \mid b
\end{align*}
\]

\[
A \Rightarrow aBBb \Rightarrow abCABb
\]
\[
A \rightarrow aBBb \mid AaA \\
B \rightarrow \epsilon \mid bCA \\
C \rightarrow AB \mid a \mid b
\]

\[A \Rightarrow aBBb \Rightarrow abCABb\]
$A \rightarrow aBBb \mid AaA$
$B \rightarrow \epsilon \mid bCA$
$C \rightarrow AB \mid a \mid b$

$A \Rightarrow aBBb \Rightarrow abCABb$
$A \rightarrow aBBb \mid AaA$

$B \rightarrow \varepsilon \mid bCA$

$C \rightarrow AB \mid a \mid b$

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb$
Derivation Tree

\[
A \rightarrow aBBb \mid AaA \\
B \rightarrow \varepsilon \mid bCA \\
C \rightarrow AB \mid a \mid b
\]

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb
\]
Derivation Tree

\[
A \rightarrow aBBb | AaA \\
B \rightarrow \varepsilon | bCA \\
C \rightarrow AB | a | b
\]

\[A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBbBb\]
A → aBBb | AaA
B → ε | bCA
C → AB | a | b

A ⇒ aBBb ⇒ abCABb ⇒ abCaBbBb ⇒ abCaBbBb
\[ A \rightarrow aBBb \mid AaA \\
B \rightarrow \epsilon \mid bCA \\
C \rightarrow AB \mid a \mid b \]

\[ A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abC aBBbBb \Rightarrow abCaBbBb \]
Derivation Tree

\[
\begin{align*}
A & \rightarrow aBBb \mid AaA \\
B & \rightarrow \varepsilon \mid bCA \\
C & \rightarrow AB \mid a \mid b
\end{align*}
\]

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abC_aBbBb
\]
Derivation Tree

\[
A \rightarrow aBBb \mid AaA \\
B \rightarrow \varepsilon \mid bCA \\
C \rightarrow AB \mid a \mid b
\]

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abBaBbBb
\]
Derivation Tree

$A \rightarrow aBBb \mid AaA$
$B \rightarrow \epsilon \mid bCA$
$C \rightarrow AB \mid a \mid b$

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb$
A → aBBb | AaA
B → ε | bCA
C → AB | a | b

A ⇒ aBBb ⇒ abCABb ⇒ abCaBBbBb ⇒ abCaBbBb ⇒ abbaBb

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019  167 / 490
Derivation Tree

\[
A \rightarrow aBBb \mid AaA \\
B \rightarrow \varepsilon \mid bCA \\
C \rightarrow AB \mid a \mid b
\]

A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb
A → aBBb | AaA
B → ε | bCA
C → AB | a | b

A ⇒ aBBb ⇒ abCABb ⇒ abCaBBbBb ⇒ abCaBbBb ⇒ abbaBbBb ⇒ abbaBbb
Derivation Tree

A → aBBb | AaA
B → ε | bCA
C → AB | a | b

A ⇒ aBBb ⇒ abCABb ⇒ abCaBBbBb ⇒ abCaBbBb ⇒ abbaBbBb ⇒ abba_B_bb
A → aBBb | AaA
B → ε | bCA
C → AB | a | b

A ⇒ aBBb ⇒ abCABb ⇒ abCaBBbBb ⇒ abCaBbBb ⇒ abbaBbBb ⇒ abbaBbBB ⇒ abbabb
Derivation Tree

\[ A \rightarrow aBBb \mid AaA \]
\[ B \rightarrow \varepsilon \mid bCA \]
\[ C \rightarrow AB \mid a \mid b \]

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb \Rightarrow abbabb
\]
For each derivation there is some deriva-
tion tree:

- Nodes of the tree are labelled with terminals and nonterminals.
- The root of the tree is labelled with the initial nonterminal.
- The leafs of the tree are labelled with terminals or with symbols $\varepsilon$.
- The remaining nodes of the tree are labelled with nonterminals.
- If a node is labelled with some nonterminal $A$ then its children are labelled with the symbols from the right-hand side of some rewriting rule $A \rightarrow \alpha$. 
Left and Right Derivation

\[ E \rightarrow E + E \mid E \ast E \mid (E) \mid a \]

A **left derivation** is a derivation where in every step we always replace the leftmost nonterminal.

\[
E \Rightarrow E + E \Rightarrow E \ast E + E \Rightarrow a \ast E + E \Rightarrow a \ast a + E \Rightarrow a \ast a + a
\]

A **right derivation** is a derivation where in every step we always replace the rightmost nonterminal.

\[
E \Rightarrow E + E \Rightarrow E + a \Rightarrow E \ast E + a \Rightarrow E \ast a + a \Rightarrow a \ast a + a
\]

A derivation need not be left or right:

\[
E \Rightarrow E + E \Rightarrow E \ast E + E \Rightarrow E \ast a + E \Rightarrow E \ast a + a \Rightarrow a \ast a + a
\]
There can be several different derivations corresponding to one derivation tree.

For every derivation tree, there is exactly one left and exactly one right derivation corresponding to the tree.
Grammars \( G_1 \) and \( G_2 \) are equivalent if they generate the same language, i.e., if \( L(G_1) = L(G_2) \).

**Remark:** The problem of equivalence of context-free grammars is algorithmically undecidable. It can be shown that it is not possible to construct an algorithm that would decide for any pair of context-free grammars if they are equivalent or not. Even the problem to decide if a grammar generates the language \( \Sigma^* \) is algorithmically undecidable.
A grammar $G$ is **ambiguous** if there is a word $w \in L(G)$ that has two different derivation trees, resp. two different left or two different right derivations.

**Example:**

\[
\begin{align*}
E & \Rightarrow E + E \\
E & \Rightarrow E * E + E \\
E & \Rightarrow a * E + E \\
a & \Rightarrow a * a + E \\
a & \Rightarrow a * a + a \\
\end{align*}
\]
Sometimes it is possible to replace an ambiguous grammar with a grammar generating the same language but which is not ambiguous.

**Example:** A grammar

\[
E \rightarrow E + E \mid E \ast E \mid (E) \mid a
\]

can be replaced with the equivalent grammar

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

**Remark:** If there is no unambiguous grammar equivalent to a given ambiguous grammar, we say it is *inherently ambiguous*. 
Definition

A language $L$ is context-free if there exists some context-free grammar $G$ such that $L = L(G)$.

The class of context-free languages is closed with respect to:

- concatenation
- union
- iteration

The class of context-free languages is not closed with respect to:

- complement
- intersection
Context-Free Languages

We have two grammars $G_1 = (\Pi_1, \Sigma, S_1, P_1)$ and $G_2 = (\Pi_2, \Sigma, S_2, P_2)$, and can assume that $\Pi_1 \cap \Pi_2 = \emptyset$ and $S \notin \Pi_1 \cup \Pi_2$.

- Grammar $G$ such that $L(G) = L(G_1)L(G_2)$:
  $$G = (\Pi_1 \cup \Pi_2 \cup \{S\}, \Sigma, S, P_1 \cup P_2 \cup \{S \rightarrow S_1S_2\})$$

- Grammar $G$ such that $L(G) = L(G_1) \cup L(G_2)$:
  $$G = (\Pi_1 \cup \Pi_2 \cup \{S\}, \Sigma, S, P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\})$$

- Grammar $G$ such that $L(G) = L(G_1)^*$:
  $$G = (\Pi_1 \cup \{S\}, \Sigma, S, P_1 \cup \{S \rightarrow \varepsilon, S \rightarrow S_1S\})$$
Example: The construction of a context-free grammar for regular expression \(((a + b) \cdot b)^*\):
Example: The construction of a context-free grammar for regular expression \(((a + b) \cdot b)^*\):
Example: The construction of a context-free grammar for regular expression \(((a + b) \cdot b)^*\):
Example: The construction of a context-free grammar for regular expression 
\(((a + b) \cdot b)^*\):

\[
\begin{align*}
S_3 & \rightarrow S_1 \mid S_2 \\
S_2 & \rightarrow b \\
S_1 & \rightarrow a
\end{align*}
\]
**Example:** The construction of a context-free grammar for regular expression \(((a + b) \cdot b)^*\):
Example: The construction of a context-free grammar for regular expression \(((a + b) \cdot b)^*\):

\[
S_5 \rightarrow \varepsilon \mid S_4 S_5 \\
S_4 \rightarrow S_3 S_2 \\
S_3 \rightarrow S_1 \mid S_2 \\
S_2 \rightarrow b \\
S_1 \rightarrow a
\]
Example:
A Context-Free Grammar for a Finite Automaton

Example:

\[ S \rightarrow A \mid C \]
A Context-Free Grammar for a Finite Automaton

Example:

\[
S \rightarrow A \mid C \\
A \rightarrow aB \mid aC \mid bA \\
B \rightarrow aD \mid bE \\
C \rightarrow bD \\
D \rightarrow bC \mid bE \mid A \\
E \rightarrow bE
\]
A Context-Free Grammar for a Finite Automaton

Example:

\[
S \rightarrow A \mid C
\]

\[
A \rightarrow aB \mid aC \mid bA
\]

\[
B \rightarrow aD \mid bE
\]

\[
C \rightarrow bD
\]

\[
D \rightarrow bC \mid bE \mid A
\]

\[
E \rightarrow bE
\]

\[
A \rightarrow \varepsilon
\]

\[
E \rightarrow \varepsilon
\]
Example:

Alternative construction:
A Context-Free Grammar for a Finite Automaton

Example:

Alternative construction:

\[ S \rightarrow A \mid E \]
A Context-Free Grammar for a Finite Automaton

Example:

Alternative construction:

\[
S \rightarrow A \mid E \\
A \rightarrow Ab \mid D \\
B \rightarrow Aa \\
C \rightarrow Aa \mid Db \\
D \rightarrow Ba \mid Cb \\
E \rightarrow Bb \mid Db \mid Eb
\]
Example:

Alternative construction:

\[
\begin{align*}
S & \rightarrow A \mid E \\
A & \rightarrow Ab \mid D \\
B & \rightarrow Aa \\
C & \rightarrow Aa \mid Db \\
D & \rightarrow Ba \mid Cb \\
E & \rightarrow Bb \mid Db \mid Eb \\
A & \rightarrow \varepsilon \\
C & \rightarrow \varepsilon
\end{align*}
\]
Regular grammars

Definition
A grammar $G = (\Pi, \Sigma, S, P)$ is **right regular** if all rules in $P$ are of the following forms (where $A, B \in \Pi$ and $a \in \Sigma$):

- $A \rightarrow B$
- $A \rightarrow aB$
- $A \rightarrow \varepsilon$

Definition
A grammar $G = (\Pi, \Sigma, S, P)$ is **left regular** if all rules in $P$ are of the following forms (where $A, B \in \Pi$, $a \in \Sigma$):

- $A \rightarrow B$
- $A \rightarrow Ba$
- $A \rightarrow \varepsilon$
Regular grammars

Definition
A grammar $G$ is **regular** if it right regular or left regular.

**Remark:** Sometimes a slightly more general definition of right (resp. left) regular grammars is given, allowing all rules of the following forms:

- $A \rightarrow wB$ (resp. $A \rightarrow Bw$)
- $A \rightarrow w$

where $A, B \in \Pi$ and $w \in \Sigma^*$.

Such rules can be easily “decomposed” into rules of the form in the previous definition.

**Example:** Rule $A \rightarrow a bbB$ can be replaced with rules

$$A \rightarrow a Z_1 \quad Z_1 \rightarrow b Z_2 \quad Z_2 \rightarrow b B$$

where $Z_1, Z_2$ are new nonterminals, not used anywhere else in the grammar.
Proposition

For every regular language $L$ there is a left regular grammar $G$ such that $L(G) = L$ and a right regular grammar $G'$ such that $L(G') = L$.

Proposition

For every regular grammar $G$ there is a finite automaton $A$ such that $L(A) = L(G)$. 
Definition

A context-free grammar $G = (\Pi, \Sigma, S, P)$ is reduced if for every $A \in \Pi$:

- there are some $u, v \in \Sigma^*$ such that $S \Rightarrow^* uAv$.
- there is some $w \in \Sigma^*$ such that $A \Rightarrow^* w$, and

Remark: Obviously, if $S \Rightarrow^* uAv$ and $A \Rightarrow^* w$ where $u, v, w \in \Sigma^*$, then $S \Rightarrow^* uwv$, and so $A$ is used in some derivation of a word from $\Sigma^*$.

On the other hand, if $A$ is used in some derivation $S \Rightarrow^* z$ of a word $z \in \Sigma^*$, then $z$ can be divided into parts $u, v, w$ such that $z = uwv$ and $S \Rightarrow^* uAv$ and $A \Rightarrow^* w$. 
Reduction of a Context-Free Grammar

Obviously, every $A \in \Pi$ with the property that

- there are no $u, v \in \Sigma^*$ such that $S \Rightarrow^* uAv$, or
- there is no $w \in \Sigma^*$ such that $S \Rightarrow^* w$,

can be safely removed from the grammar (together with all rules where it occurs) without affecting the generated language.
Reduction of a Context-Free Grammar

An algorithm that for a given CFG $G$ constructs an equivalent reduced grammar:

1. Construct the set $\mathcal{T}$ of all nonterminals that can generate a terminal word:

   $$\mathcal{T} = \{ A \in \Pi \mid (\exists w \in \Sigma^*)(A \Rightarrow^* w) \}$$

2. Remove from $G$ all nonterminals from the set $\Pi - \mathcal{T}$ together with all rules where they occur. Denote the resulting grammar $G' = (\Pi', \Sigma, S, P)$.

3. Construct the set $\mathcal{D}$ of all nonterminals that can be “reached” from the initial nonterminal $S$:

   $$\mathcal{D} = \{ A \in \Pi' \mid (\exists \alpha, \beta \in (\Pi' \cup \Sigma)^*)(S \Rightarrow^* \alpha A \beta) \}$$

4. Remove from $G'$ all nonterminals from the set $\Pi' - \mathcal{D}$ together with all rules where they occur. The resulting grammar $G''$ is the result of the whole algorithm.
Example:

\[
\begin{align*}
S & \rightarrow AC \mid B \\
A & \rightarrow aC \mid AbA \\
B & \rightarrow Ba \mid BbA \mid DB \\
C & \rightarrow aa \mid aBC \\
D & \rightarrow aA \mid \varepsilon
\end{align*}
\]
Reduction of a Context-Free Grammar

Example:

\[ T_0 = \{C, D\} \]

\[
\begin{align*}
S & \rightarrow AC \mid B \\
A & \rightarrow aC \mid AbA \\
B & \rightarrow Ba \mid BbA \mid DB \\
C & \rightarrow aa \mid aBC \\
D & \rightarrow aA \mid \varepsilon
\end{align*}
\]
Example:

\[ \mathcal{T}_0 = \{C, D\} \]
\[ \mathcal{T}_1 = \{C, D, A\} \]

\[
S \rightarrow AC \mid B \\
A \rightarrow aC \mid AbA \\
B \rightarrow Ba \mid BbA \mid DB \\
C \rightarrow aa \mid aBC \\
D \rightarrow aA \mid \varepsilon
\]
Example:

\[ T_0 = \{C, D\} \]
\[ T_1 = \{C, D, A\} \]
\[ T_2 = \{C, D, A, S\} \]

\[ S \rightarrow AC \mid B \]
\[ A \rightarrow aC \mid AbA \]
\[ B \rightarrow Ba \mid BbA \mid DB \]
\[ C \rightarrow aa \mid aBC \]
\[ D \rightarrow aA \mid \varepsilon \]
Example:

\[ S \rightarrow AC \mid B \]
\[ A \rightarrow aC \mid AbA \]
\[ B \rightarrow Ba \mid BbA \mid DB \]
\[ C \rightarrow aa \mid aBC \]
\[ D \rightarrow aA \mid \varepsilon \]

\[ T_0 = \{ C, D \} \]
\[ T_1 = \{ C, D, A \} \]
\[ T_2 = \{ C, D, A, S \} \]

\[ T = \{ C, D, A, S \} \]
Reduction of a Context-Free Grammar

Example:

\[ S \rightarrow AC \mid B \]
\[ A \rightarrow aC \mid AbA \]
\[ B \rightarrow Ba \mid BbA \mid DB \]
\[ C \rightarrow aa \mid aBC \]
\[ D \rightarrow aA \mid \varepsilon \]

\[ T_0 = \{C, D\} \]
\[ T_1 = \{C, D, A\} \]
\[ T_2 = \{C, D, A, S\} \]

\[ T = \{C, D, A, S\} \]

\[ S \rightarrow AC \]
\[ A \rightarrow aC \mid AbA \]
\[ C \rightarrow aa \]
\[ D \rightarrow aA \mid \varepsilon \]
Example:

\[ T_0 = \{C, D\} \]
\[ T_1 = \{C, D, A\} \]
\[ T_2 = \{C, D, A, S\} \]
\[ \mathcal{D}_0 = \{S\} \]

\[ S \rightarrow AC | B \]
\[ A \rightarrow aC | AbA \]
\[ B \rightarrow Ba | BbA | DB \]
\[ C \rightarrow aa | aBC \]
\[ D \rightarrow aA | \epsilon \]

\[ S \rightarrow AC \]
\[ A \rightarrow aC | AbA \]
\[ C \rightarrow aa \]
\[ D \rightarrow aA | \epsilon \]
Example:

\[ S \rightarrow AC \mid B \]
\[ A \rightarrow aC \mid AbA \]
\[ B \rightarrow Ba \mid BbA \mid DB \]
\[ C \rightarrow aa \mid aBC \]
\[ D \rightarrow aA \mid \varepsilon \]

\[ T_0 = \{C, D\} \]
\[ T_1 = \{C, D, A\} \]
\[ T_2 = \{C, D, A, S\} \]
\[ D_0 = \{S\} \]
\[ D_1 = \{S, A, C\} \]

\[ T = \{C, D, A, S\} \]

\[ S \rightarrow AC \]
\[ A \rightarrow aC \mid AbA \]
\[ C \rightarrow aa \]
\[ D \rightarrow aA \mid \varepsilon \]
Example:

\[
\begin{align*}
S & \rightarrow AC \mid B \\
A & \rightarrow aC \mid AbA \\
B & \rightarrow Ba \mid BbA \mid DB \\
C & \rightarrow aa \mid aBC \\
D & \rightarrow aA \mid \varepsilon
\end{align*}
\]

\[
\begin{align*}
\mathcal{T}_0 & = \{C, D\} \\
\mathcal{T}_1 & = \{C, D, A\} \\
\mathcal{T}_2 & = \{C, D, A, S\} \\
\mathcal{T} & = \{C, D, A, S\} \\
\mathcal{D}_0 & = \{S\} \\
\mathcal{D}_1 & = \{S, A, C\} \\
\mathcal{D} & = \{S, A, C\}
\end{align*}
\]
Example:

\[ T_0 = \{ C, D \} \]
\[ T_1 = \{ C, D, A \} \]
\[ T_2 = \{ C, D, A, S \} \]
\[ S \rightarrow AC \mid B \]
\[ A \rightarrow aC \mid AbA \]
\[ B \rightarrow Ba \mid BbA \mid DB \]
\[ C \rightarrow aa \mid aBC \]
\[ D \rightarrow aA \mid \varepsilon \]

\[ D_0 = \{ S \} \]
\[ D_1 = \{ S, A, C \} \]

\[ T = \{ C, D, A, S \} \]
\[ D = \{ S, A, C \} \]
Removing Epsilon-rules

Rules of the form $A \rightarrow \varepsilon$ are called epsilon-rules ($\varepsilon$-rules).

**Proposition**

For every context-free grammar $G$ there is a context-free grammar $G'$ without $\varepsilon$-rules such that $L(G') = L(G) - \{\varepsilon\}$.

**Proof:** Construct the set $E$ of all nonterminals that can be rewritten to $\varepsilon$, i.e.,

$$E = \{ A \in \Pi \mid A \Rightarrow^* \varepsilon \}$$

Remove all $\varepsilon$-rules and replace every other rule $A \rightarrow \alpha$ with a set of rules obtained by all possible rules of the form $A \rightarrow \alpha'$ where $\alpha'$ is obtained from $\alpha$ by possible omitting of (some) occurrences of nonterminals from $E$. 
Removing Epsilon-rules

Example:

\[ S \rightarrow ASA \mid aBC \mid b \]
\[ A \rightarrow BD \mid aAB \]
\[ B \rightarrow bB \mid \varepsilon \]
\[ C \rightarrow AaA \mid b \]
\[ D \rightarrow AD \mid BBB \mid a \]
Removing Epsilon-rules

Example:

\[ \mathcal{E}_0 = \{B\} \]

\[
S \rightarrow ASA \mid aBC \mid b \\
A \rightarrow BD \mid aAB \\
B \rightarrow bB \mid \varepsilon \\
C \rightarrow AaA \mid b \\
D \rightarrow AD \mid BBB \mid a
\]
Example:

\[ \mathcal{E}_0 = \{ B \} \]
\[ \mathcal{E}_1 = \{ B, D \} \]

\[
\begin{align*}
S & \rightarrow ASA \mid aBC \mid b \\
A & \rightarrow BD \mid aAB \\
B & \rightarrow bB \mid \varepsilon \\
C & \rightarrow AaA \mid b \\
D & \rightarrow AD \mid BBB \mid a
\end{align*}
\]
Removing Epsilon-rules

Example:

\[ \begin{align*}
\mathcal{E}_0 &= \{B\} \\
\mathcal{E}_1 &= \{B, D\} \\
\mathcal{E}_2 &= \{B, D, A\}
\end{align*} \]

\[ \begin{align*}
S &\rightarrow ASA \mid aBC \mid b \\
A &\rightarrow BD \mid aAB \\
B &\rightarrow bB \mid \varepsilon \\
C &\rightarrow AaA \mid b \\
D &\rightarrow AD \mid BBB \mid a
\end{align*} \]
Removing Epsilon-rules

Example:

\[\begin{align*}
\mathcal{E}_0 &= \{B\} \\
\mathcal{E}_1 &= \{B, D\} \\
\mathcal{E}_2 &= \{B, D, A\}
\end{align*}\]

\[\begin{align*}
S &\rightarrow ASA | aBC | b \\
A &\rightarrow BD | aAB \\
B &\rightarrow bB | \varepsilon \\
C &\rightarrow AaA | b \\
D &\rightarrow AD | BBB | a
\end{align*}\]
Removing Epsilon-rules

Example:

\[\begin{align*}
\mathcal{E}_0 &= \{B\} \\
\mathcal{E}_1 &= \{B, D\} \\
\mathcal{E}_2 &= \{B, D, A\}
\end{align*}\]

\[\begin{align*}
S &\rightarrow ASA \mid aBC \mid b \\
A &\rightarrow BD \mid aAB \\
B &\rightarrow bB \mid \varepsilon \\
C &\rightarrow AaA \mid b \\
D &\rightarrow AD \mid BBB \mid a
\end{align*}\]

\[\begin{align*}
S &\rightarrow ASA \mid SA \mid AS \mid S \mid aBC \mid aC \mid b \\
A &\rightarrow BD \mid B \mid D \mid aAB \mid aB \mid aA \mid a \\
B &\rightarrow bB \mid b \\
C &\rightarrow AaA \mid aA \mid Aa \mid a \mid b \\
D &\rightarrow AD \mid D \mid A \mid BBB \mid BB \mid B \mid a
\end{align*}\]
Removing Epsilon-rules

For every context-free grammar \( G = (\Pi, \Sigma, S, P) \) there is a context-free grammar \( G' = (\Pi', \Sigma, S', P') \) such that \( L(G') = L(G) \) and either:

- \( G' \) does not contain \( \varepsilon \)-rules, or
- the only \( \varepsilon \)-rule in \( G' \) is the rule \( S' \rightarrow \varepsilon \) and \( S' \) does not occur on the right-hand side of any rule in \( G' \).
Removing Unit-rules

Rules of the form $A \rightarrow B$ where $A, B \in \Pi$ are called unit rules.

**Proposition**
For every context-free grammar $G$ there is a context-free grammar $G'$ without $\varepsilon$-rules and without unit rules such that $L(G') = L(G) - \{\varepsilon\}$.

**Proof:** Assume $G = (\Pi, \Sigma, S, P)$ does not contain $\varepsilon$-rules.
For each $A \in \Pi$ compute the set $\mathcal{N}_A$ of all nonterminals that can be obtained from $A$ by using only unit rules, i.e.,

$$\mathcal{N}_A = \{ B \in \Pi \mid A \Rightarrow^* B \}$$

Construct CFG $G' = (\Pi, \Sigma, S, P')$ where $P'$ consist of rules of the form $A \rightarrow \beta$ where $A \in \Pi$, $\beta$ is not a single nonterminal, and $(B \in \beta) \in P$ for some $B \in \mathcal{N}_A$. 
Removing Unit-rules

Example:

\[ S \rightarrow AB | C \]
\[ A \rightarrow a | bA \]
\[ B \rightarrow C | b \]
\[ C \rightarrow D | AA | AaA \]
\[ D \rightarrow B | ABb \]
Removing Unit-rules

Example:

\[ \mathcal{N}^0_S = \{S\} \]

\[ S \rightarrow AB \mid C \]
\[ A \rightarrow a \mid bA \]
\[ B \rightarrow C \mid b \]
\[ C \rightarrow D \mid AA \mid AaA \]
\[ D \rightarrow B \mid ABb \]
Removing Unit-rules

Example:

\[ N^0_S = \{ S \} \]
\[ N^1_S = \{ S, C \} \]

\[
S \rightarrow AB \mid C \\
A \rightarrow a \mid bA \\
B \rightarrow C \mid b \\
C \rightarrow D \mid AA \mid AaA \\
D \rightarrow B \mid ABb
\]
Removing Unit-rules

Example:

\[
\begin{align*}
\mathcal{N}_S^0 &= \{S\} \\
\mathcal{N}_S^1 &= \{S, C\} \\
\mathcal{N}_S^2 &= \{S, C, D\}
\end{align*}
\]

\[
\begin{align*}
S &\rightarrow AB \mid C \\
A &\rightarrow a \mid bA \\
B &\rightarrow C \mid b \\
C &\rightarrow D \mid AA \mid AaA \\
D &\rightarrow B \mid ABb
\end{align*}
\]
Removing Unit-rules

Example:

\[ \mathcal{N}_S^0 = \{S\} \]
\[ \mathcal{N}_S^1 = \{S, C\} \]
\[ \mathcal{N}_S^2 = \{S, C, D\} \]
\[ \mathcal{N}_S^3 = \{S, C, D, B\} \]

\[ S \rightarrow AB \mid C \]
\[ A \rightarrow a \mid bA \]
\[ B \rightarrow C \mid b \]
\[ C \rightarrow D \mid AA \mid AaA \]
\[ D \rightarrow B \mid ABb \]
Removing Unit-rules

Example:

\[ N^0_S = \{S\} \]
\[ N^1_S = \{S, C\} \]
\[ N^2_S = \{S, C, D\} \]
\[ N^3_S = \{S, C, D, B\} \]

\[ N^0_A = \{A\} \]

\[ S \rightarrow AB \mid C \]
\[ A \rightarrow a \mid bA \]
\[ B \rightarrow C \mid b \]
\[ C \rightarrow D \mid AA \mid AaA \]
\[ D \rightarrow B \mid ABb \]
Removing Unit-rules

Example:

\[ N^0_S = \{ S \} \]
\[ N^1_S = \{ S, C \} \]
\[ N^2_S = \{ S, C, D \} \]
\[ N^3_S = \{ S, C, D, B \} \]

\[ N^0_A = \{ A \} \]
\[ N^0_B = \{ B \} \]

\[ S \to AB \mid C \]
\[ A \to a \mid bA \]
\[ B \to C \mid b \]
\[ C \to D \mid AA \mid AaA \]
\[ D \to B \mid ABb \]
Removing Unit-rules

Example:

\[
\begin{align*}
\mathcal{N}_S^0 &= \{S\} \\
\mathcal{N}_S^1 &= \{S, C\} \\
\mathcal{N}_S^2 &= \{S, C, D\} \\
\mathcal{N}_S^3 &= \{S, C, D, B\} \\
\mathcal{N}_A^0 &= \{A\} \\
S &\rightarrow AB \mid C \\
A &\rightarrow a \mid bA \\
B &\rightarrow C \mid b \\
C &\rightarrow D \mid AA \mid AaA \\
D &\rightarrow B \mid ABb
\end{align*}
\]
Removing Unit-rules

Example:

\[ S \rightarrow AB \mid C \]
\[ A \rightarrow a \mid bA \]
\[ B \rightarrow C \mid b \]
\[ C \rightarrow D \mid AA \mid AaA \]
\[ D \rightarrow B \mid ABb \]

\[ N^0_S = \{S\} \]
\[ N^1_S = \{S, C\} \]
\[ N^2_S = \{S, C, D\} \]
\[ N^3_S = \{S, C, D, B\} \]

\[ N^0_A = \{A\} \]
\[ N^0_B = \{B\} \]
\[ N^1_B = \{B, C\} \]
\[ N^2_B = \{B, C, D\} \]
Removing Unit-rules

Example:

\[ N^0_S = \{S\} \]
\[ N^1_S = \{S, C\} \]
\[ N^2_S = \{S, C, D\} \]
\[ N^3_S = \{S, C, D, B\} \]

\[ N^0_A = \{A\} \]

\[ S \rightarrow AB \mid C \]
\[ A \rightarrow a \mid bA \]
\[ B \rightarrow C \mid b \]
\[ C \rightarrow D \mid AA \mid AaA \]
\[ D \rightarrow B \mid ABb \]

\[ N^0_B = \{B\} \]
\[ N^1_B = \{B, C\} \]
\[ N^2_B = \{B, C, D\} \]
\[ N^0_C = \{C\} \]
Removing Unit-rules

Example:

\[ \mathcal{N}_S^0 = \{S\} \]
\[ \mathcal{N}_S^1 = \{S, C\} \]
\[ \mathcal{N}_S^2 = \{S, C, D\} \]
\[ \mathcal{N}_S^3 = \{S, C, D, B\} \]

\[ \mathcal{N}_A^0 = \{A\} \]

\[ S \rightarrow AB \mid C \]
\[ A \rightarrow a \mid bA \]
\[ B \rightarrow C \mid b \]
\[ C \rightarrow D \mid AA \mid AaA \]
\[ D \rightarrow B \mid ABb \]
Removing Unit-rules

Example:

\[
\begin{align*}
\mathcal{N}_S^0 &= \{S\} \\
\mathcal{N}_S^1 &= \{S, C\} \\
\mathcal{N}_S^2 &= \{S, C, D\} \\
\mathcal{N}_S^3 &= \{S, C, D, B\} \\
\mathcal{N}_A^0 &= \{A\} \\
\mathcal{N}_B^0 &= \{B\} \\
\mathcal{N}_B^1 &= \{B, C\} \\
\mathcal{N}_B^2 &= \{B, C, D\} \\
\mathcal{N}_C^0 &= \{C\} \\
\mathcal{N}_C^1 &= \{C, D\} \\
\mathcal{N}_C^2 &= \{C, D, B\}
\end{align*}
\]

\[
\begin{align*}
S &\rightarrow AB \mid C \\
A &\rightarrow a \mid bA \\
B &\rightarrow C \mid b \\
C &\rightarrow D \mid AA \mid AaA \\
D &\rightarrow B \mid ABb
\end{align*}
\]
Removing Unit-rules

Example:

\[ N_0^S = \{S\} \]
\[ N_1^S = \{S, C\} \]
\[ N_2^S = \{S, C, D\} \]
\[ N_3^S = \{S, C, D, B\} \]

\[ N_0^A = \{A\} \]
\[ N_0^B = \{B\} \]
\[ N_1^B = \{B, C\} \]
\[ N_2^B = \{B, C, D\} \]
\[ N_0^C = \{C\} \]
\[ N_1^C = \{C, D\} \]
\[ N_2^C = \{C, D, B\} \]
\[ N_0^D = \{D\} \]
Removing Unit-rules

**Example:**

\[
\begin{align*}
\mathcal{N}^0_S &= \{S\} \\
\mathcal{N}^1_S &= \{S, C\} \\
\mathcal{N}^2_S &= \{S, C, D\} \\
\mathcal{N}^3_S &= \{S, C, D, B\} \\
\mathcal{N}^0_A &= \{A\} \\
\mathcal{N}^0_B &= \{B\} \\
\mathcal{N}^1_B &= \{B, C\} \\
\mathcal{N}^2_B &= \{B, C, D\} \\
\mathcal{N}^0_C &= \{C\} \\
\mathcal{N}^1_C &= \{C, D\} \\
\mathcal{N}^2_C &= \{C, D, B\} \\
\mathcal{N}^0_D &= \{D\} \\
\mathcal{N}^1_D &= \{D, B\}
\end{align*}
\]

\[
\begin{align*}
S &\rightarrow AB \mid C \\
A &\rightarrow a \mid bA \\
B &\rightarrow C \mid b \\
C &\rightarrow D \mid AA \mid AaA \\
D &\rightarrow B \mid ABb
\end{align*}
\]
Removing Unit-rules

Example:

\[ S \rightarrow AB \mid C \]
\[ A \rightarrow a \mid bA \]
\[ B \rightarrow C \mid b \]
\[ C \rightarrow D \mid AA \mid AaA \]
\[ D \rightarrow B \mid ABb \]

\[ \mathcal{N}_S^0 = \{ S \} \]
\[ \mathcal{N}_S^1 = \{ S, C \} \]
\[ \mathcal{N}_S^2 = \{ S, C, D \} \]
\[ \mathcal{N}_S^3 = \{ S, C, D, B \} \]

\[ \mathcal{N}_A^0 = \{ A \} \]
\[ \mathcal{N}_B^0 = \{ B \} \]
\[ \mathcal{N}_B^1 = \{ B, C \} \]
\[ \mathcal{N}_B^2 = \{ B, C, D \} \]

\[ \mathcal{N}_C^0 = \{ C \} \]
\[ \mathcal{N}_C^1 = \{ C, D \} \]
\[ \mathcal{N}_C^2 = \{ C, D, B \} \]

\[ \mathcal{N}_D^0 = \{ D \} \]
\[ \mathcal{N}_D^1 = \{ D, B \} \]
\[ \mathcal{N}_D^2 = \{ D, B, C \} \]
Removing Unit-rules

Example:

\[
\begin{align*}
N_S^0 &= \{S\} \\
N_S^1 &= \{S, C\} \\
N_S^2 &= \{S, C, D\} \\
N_S^3 &= \{S, C, D, B\} \\
N_A^0 &= \{A\} \\
N_B^0 &= \{B\} \\
N_B^1 &= \{B, C\} \\
N_B^2 &= \{B, C, D\} \\
N_C^0 &= \{C\} \\
N_C^1 &= \{C, D\} \\
N_C^2 &= \{C, D, B\} \\
N_D^0 &= \{D\} \\
N_D^1 &= \{D, B\} \\
N_D^2 &= \{D, B, C\}
\end{align*}
\]
Removing Unit-rules

Example:

\[ S \rightarrow AB \mid C \]
\[ A \rightarrow a \mid bA \]
\[ B \rightarrow C \mid b \]
\[ C \rightarrow D \mid AA \mid AaA \]
\[ D \rightarrow B \mid ABb \]

\[ N_S^0 = \{S\} \]
\[ N_S^1 = \{S, C\} \]
\[ N_S^2 = \{S, C, D\} \]
\[ N_S^3 = \{S, C, D, B\} \]

\[ N_A^0 = \{A\} \]
\[ N_B^0 = \{B\} \]
\[ N_B^1 = \{B, C\} \]
\[ N_B^2 = \{B, C, D\} \]
\[ N_C^0 = \{C\} \]
\[ N_C^1 = \{C, D\} \]
\[ N_C^2 = \{C, D, B\} \]
\[ N_D^0 = \{D\} \]
\[ N_D^1 = \{D, B\} \]
\[ N_D^2 = \{D, B, C\} \]

\[ N_S = \{S, C, D, B\} \]
\[ N_A = \{A\} \]
\[ N_B = \{B, C, D\} \]
\[ N_C = \{C, D, B\} \]
\[ N_D = \{D, B, C\} \]

\[ S \rightarrow AB \mid AA \mid AaA \mid ABb \mid b \]
\[ A \rightarrow a \mid bA \]
\[ B \rightarrow b \mid AA \mid AaA \mid ABb \]
\[ C \rightarrow AA \mid AaA \mid ABb \mid b \]
\[ D \rightarrow ABb \mid b \mid AA \mid AaA \]
A context-free grammar is in **Chomsky normal form** if every rule is of the following forms:

- $A \rightarrow BC$
- $A \rightarrow a$

where $a$ is any terminal and $A$, $B$, and $C$ are any nonterminals — except that $B$ and $C$ may not be the initial nonterminal. In addition we permit the rule $S \rightarrow \varepsilon$, where $S$ is the initial nonterminal.
Proposition

For every context-free grammar $G$ there is an equivalent context-free grammar $G'$ in Chomsky normal form.

Proof: Perform the following transformations on $G$:

1. Decompose each rule $A \rightarrow \alpha$ where $|\alpha| \geq 2$ into a sequence of rules where each right-hand side has length 2.

2. Remove $\epsilon$-rules.

3. Remove unit rules.

4. For each terminal $a$ occurring on the right-hand size of some rule $A \rightarrow \alpha$ where $|\alpha| = 2$ introduce a new nonterminal $N_a$, replace occurrences of $a$ on such right-hand sides with $N_a$, and add $N_a \rightarrow a$ as a new rule.
Chomsky Normal Form

Example:

\[ S \rightarrow ASA \mid aB \]
\[ A \rightarrow B \mid S \]
\[ B \rightarrow b \mid \varepsilon \]
Chomsky Normal Form

Example:

\[
\begin{align*}
S & \rightarrow ASA | aB \\
A & \rightarrow B | S \\
B & \rightarrow b | \varepsilon \\
\end{align*}
\]

Step 1:

\[
\begin{align*}
S & \rightarrow AZ | aB \\
Z & \rightarrow SA \\
A & \rightarrow B | S \\
B & \rightarrow b | \varepsilon \\
\end{align*}
\]
Chomsky Normal Form

Example:

\[ S \rightarrow ASA \mid aB \]
\[ A \rightarrow B \mid S \]
\[ B \rightarrow b \mid \varepsilon \]

Step 1:

\[ S \rightarrow AZ \mid aB \]
\[ Z \rightarrow SA \]
\[ A \rightarrow B \mid S \]
\[ B \rightarrow b \mid \varepsilon \]

Step 2:

\[ \mathcal{E} = \{ B, A \} \]
Chomsky Normal Form

Example:

Step 1:

\[
\begin{align*}
S & \rightarrow ASA | aB \\
A & \rightarrow B | S \\
B & \rightarrow b | \varepsilon
\end{align*}
\]

Step 2:

\[
\mathcal{E} = \{B, A\}
\]

\[
\begin{align*}
S_0 & \rightarrow S \\
S & \rightarrow AZ | Z | aB | a \\
Z & \rightarrow SA | S \\
A & \rightarrow B | S \\
B & \rightarrow b
\end{align*}
\]
Chomsky Normal Form

Example:

Step 1:

\[
\begin{align*}
S & \rightarrow ASA \mid aB \\
A & \rightarrow B \mid S \\
B & \rightarrow b \mid \varepsilon
\end{align*}
\]

Step 2:

\[
\begin{align*}
E & = \{B, A\} \\
S_0 & \rightarrow S \\
S & \rightarrow AZ \mid Z \mid aB \mid a \\
Z & \rightarrow SA \mid S \\
A & \rightarrow B \mid S \\
B & \rightarrow b
\end{align*}
\]

Step 3:

\[
\begin{align*}
\mathcal{N}_{S_0} & = \{S_0, S, Z\} \\
\mathcal{N}_S & = \{S, Z\} \\
\mathcal{N}_Z & = \{Z, S\} \\
\mathcal{N}_A & = \{A, B, S, Z\} \\
\mathcal{N}_B & = \{B\}
\end{align*}
\]
Chomsky Normal Form

Example:

Step 1:

\[
\begin{align*}
S & \rightarrow ASA \mid aB \\
A & \rightarrow B \mid S \\
B & \rightarrow b \mid \varepsilon
\end{align*}
\]

Step 2:

\[
\begin{align*}
E &= \{B, A\} \\
S_0 &\rightarrow S \\
S &\rightarrow AZ \mid Z \mid aB \mid a \\
Z &\rightarrow SA \mid S \\
A &\rightarrow B \mid S \\
B &\rightarrow b
\end{align*}
\]

Step 3:

\[
\begin{align*}
\mathcal{N}_{S_0} &= \{S_0, S, Z\} \\
\mathcal{N}_S &= \{S, Z\} \\
\mathcal{N}_Z &= \{Z, S\} \\
\mathcal{N}_A &= \{A, B, S, Z\} \\
\mathcal{N}_B &= \{B\}
\end{align*}
\]

\[
\begin{align*}
S_0 &\rightarrow AZ \mid aB \mid a \mid SA \\
S &\rightarrow AZ \mid aB \mid a \mid SA \\
Z &\rightarrow SA \mid AZ \mid aB \mid a \\
A &\rightarrow b \mid AZ \mid aB \mid a \mid SA \\
B &\rightarrow b
\end{align*}
\]
Chomsky Normal Form

Example:

\[ S \rightarrow ASA | aB \]
\[ A \rightarrow B | S \]
\[ B \rightarrow b | \varepsilon \]

Step 1:

\[ S \rightarrow AZ | aB \]
\[ Z \rightarrow SA \]
\[ A \rightarrow B | S \]
\[ B \rightarrow b | \varepsilon \]

Step 2:

\[ \mathcal{E} = \{B, A\} \]
\[ S_0 \rightarrow S \]
\[ S \rightarrow AZ | Z | aB | a \]
\[ Z \rightarrow SA | S \]
\[ A \rightarrow B | S \]
\[ B \rightarrow b \]

Step 3:

\[ \mathcal{N}_{S_0} = \{S_0, S, Z\} \]
\[ \mathcal{N}_S = \{S, Z\} \]
\[ \mathcal{N}_Z = \{Z, S\} \]
\[ \mathcal{N}_A = \{A, B, S, Z\} \]
\[ \mathcal{N}_B = \{B\} \]

Step 4:

\[ S_0 \rightarrow AZ | aB | a | SA \]
\[ S \rightarrow AZ | aB | a | SA \]
\[ Z \rightarrow SA | AZ | aB | a \]
\[ A \rightarrow b | AZ | aB | a | SA \]
\[ B \rightarrow b \]

\[ Y \rightarrow a \]
Lexical and Syntactic Analysis — an example

Example: We would like to recognize a language of arithmetic expressions containing expressions such as:

\[ 34 \quad x+1 \quad -x \times 2 + 128 \times (y - z / 3) \]

- The expressions can contain number constants — sequences of digits \(0, 1, \ldots, 9\).
- The expressions can contain names of variables — sequences consisting of letters, digits, and symbol ‘_’, which do not start with a digit.
- The expressions can contain basic arithmetic operations — ‘+’, ‘-’, ‘*’, ‘/’, and unary ‘-’.
- It is possible to use parentheses — ‘(’ and ‘)’, and to use a standard priority of arithmetic operations.
Lexical and Syntactic Analysis — an example

- **Input:** a sequence of characters (e.g., a string, a text file, etc.)
- **Output:** an abstract syntax tree representing the structure of a given expression, or an information about a syntax error in the expression
Lexical and Syntactic Analysis — an example

Construction of an **abstract syntax tree**:

- An enumerated type representing binary arithmetic operations:
  ```
  enum Bin_op { Add, Sub, Mul, Div }
  ```

- An enumerated type representing unary arithmetic operations:
  ```
  enum Un_op { Un_minus }
  ```

- Functions for creation of different kinds of nodes of an abstract syntax tree:
  - **MK-VAR**(ident) — creates a leaf representing a variable
  - **MK-NUM**(num) — creates a leaf representing a number constant
  - **MK-UNARY**(op, e) — creates a node with one child e, on which a unary operation op (of type Un_op) is applied
  - **MK-BINARY**(op, e1, e2) — creates a node with two children e1 and e2, on which a binary operation op (of type Bin_op) is applied
Enumerated type \textit{Token\_kind} representing different kinds of \textit{tokens}:

- \texttt{T\_EOF} — the end of input
- \texttt{T\_Ident} — identifier
- \texttt{T\_Number} — number constant
- \texttt{T\_LParen} — “(”
- \texttt{T\_RParen} — “)”
- \texttt{T\_Plus} — “+”
- \texttt{T\_Minus} — “-”
- \texttt{T\_Star} — “*”
- \texttt{T\_Slash} — “/”
Variable $c$: a currently processed character (resp. a special value $\langle eo\rangle$ representing the end of input):

- at the beginning, the first character in the input is read to variable $c$
- function NEXT-CHAR() returns a next character from the input

Some helper functions:

- **ERROR()** — outputs an information about a syntax error and aborts the processing of the expression
- **is-ident-start-char(c)** — tests whether $c$ is a character that can occur at the beginning of an identifier
- **is-ident-normal-char(c)** — tests whether $c$ is a character that can occur in an identifier (on other positions except beginning)
- **is-digit(c)** — tests whether $c$ is a digit
Lexical Analysis

Some other helper functions:

- $\text{CREATE-IDENT}(s)$ — creates an identifier from a given string $s$
- $\text{CREATE-NUMBER}(s)$ — creates a number from a given string $s$

Auxiliary variables:

- $\textit{last-ident}$ — the last processed identifier
- $\textit{last-num}$ — the last processed number constant

Function $\text{NEXT-TOKEN}()$ — the main part of the lexical analyser, it returns the following token from the input
NEXT-TOKEN() :
begin
   while c ∈ {“ “, “\t”} do
      c := NEXT-CHAR();
   end
   if c == ⟨eof⟩ then
      return T_EOF
   else
      switch c do
         case “(” : do c := NEXT-CHAR(); return T_LParen
         case “)” : do c := NEXT-CHAR(); return T_RParen
         case “+” : do c := NEXT-CHAR(); return T_Plus
         case “–” : do c := NEXT-CHAR(); return T_Minus
         case “*” : do c := NEXT-CHAR(); return T_Star
         case “/” : do c := NEXT-CHAR(); return T_Slash
         otherwise do
            if is-ident-start-char(c) then
               return SCAN-IDENT()
            else if is-digit(c) then
               return SCAN-NUMBER()
            else
               ERROR()
         end
      end
   end
end
Lexical Analysis

SCAN-IDENT ():

begin

  s := c
  c := NEXT-CHAR()

  while is-ident-normal-char(c) do
    s := s · c
    c := NEXT-CHAR()

  end

last-ident := CREATE-IDENT(s)
return T_Ident

end
Lexical Analysis

1 SCAN-NUMBER():
2 begin
3 \( s := c \)
4 \( c := \text{NEXT-CHAR}() \)
5 while is-digit\( (c) \) do
6 \( s := s \cdot c \)
7 \( c := \text{NEXT-CHAR}() \)
8 end
9 \( \text{last-num} := \text{CREATE-NUMBER}(s) \)
10 return T_Number
11 end
Variable $t$:
- the last processed token

A helper function:
- **INIT-SCANNER()**:
  - initializes the lexical analyser
  - reads the first character from the input into variable $c$
The context-free grammar for the given language:

\[
S \rightarrow E \langle \text{eof} \rangle \\
E \rightarrow T G \\
G \rightarrow \varepsilon \mid A T G \\
A \rightarrow + \mid - \\
T \rightarrow F U \\
U \rightarrow \varepsilon \mid M F U \\
M \rightarrow \ast \mid / \\
F \rightarrow - F \mid ( E ) \mid \langle \text{ident} \rangle \mid \langle \text{num} \rangle 
\]
One of the often used methods of syntactic analysis is **recursive descent**:

- For each nonterminal there is a corresponding function — the function corresponding to nonterminal $A$ implements all rules with nonterminal $A$ on the left-hand side.

- In a given function, the next token is used to select between corresponding rules.

- Instructions in the body of a function correspond to processing of right-hand sides of the rules:
  - an occurrence of nonterminal $B$ — the function corresponding to nonterminal $B$ is called
  - an occurrence of terminal $a$ — it is checked that the following token corresponds to terminal $a$, when it does, the next token is read, otherwise an error is reported
$S \rightarrow E \langle\text{eof}\rangle$

---

1 Parse():
2 begin
3  INIT-SCANNER()
4  $t := $ NEXT-TOKEN()
5  $e := $ Parse-E()
6  if $t == $ T_EOF then return $e$
7  else ERROR()
8 end
\[ E \rightarrow T \ G \]

---

1. Parse-E():
2. begin
3. \( e1 := \text{Parse-T}() \)
4. return Parse-G(e1)
5. end

---

\[ G \rightarrow \varepsilon \mid A \ T \ G \]

---

1. Parse-G(e1):
2. begin
3. if \( t \in \{ T\_Plus, T\_Minus \} \) then
4. \( op := \text{Parse-A}() \)
5. \( e2 := \text{Parse-T}() \)
6. return Parse-G(mk-binary(op, e1, e2))
7. else return e1
8. end
\[ T \rightarrow F U \]

**Parse-T():**
```plaintext
begin
  e1 := Parse-F()
  return Parse-U(e1)
end
```

\[ U \rightarrow \epsilon \mid MFU \]

**Parse-U(e1):**
```plaintext
begin
  if \( t \in \{\text{T-Star, T-Slash}\} \) then
    op := Parse-M()
    e2 := Parse-F()
    return Parse-U(mk-binary(op, e1, e2))
  else return e1
end
```
Syntactic Analysis

\[ A \rightarrow + \mid - \]

---

1 \textbf{PARSE-A}():
2 \textbf{begin}
3 \hspace{1em} \textbf{if} \ t == \text{\texttt{T\_Plus}} \textbf{then}
4 \hspace{2.5em} t := \text{NEXT\_TOKEN}()
5 \hspace{2.5em} \textbf{return} \ Add
6 \hspace{1em} \textbf{else if} \ t == \text{\texttt{T\_Minus}} \textbf{then}
7 \hspace{2.5em} t := \text{NEXT\_TOKEN}()
8 \hspace{2.5em} \textbf{return} \ Sub
9 \hspace{1em} \textbf{else} \ \text{ERROR}()
10 \textbf{end}
\[ M \rightarrow * \mid / \]

---

1 `PARSE-M()`:
2 `begin`
3 `if t == T_Star then`
4 \[ t := \text{NEXT-TOKEN}() \]
5 `return Mul`
6 `else if t == T_Slash then`
7 \[ t := \text{NEXT-TOKEN}() \]
8 `return Div`
9 `else` `ERROR()`
10 `end`
\[ F \to - F \]
\[ | (E) \]
\[ | \langle \text{ident} \rangle \]
\[ | \langle \text{num} \rangle \]

\begin{verbatim}
PARSE-F():
begin
    switch t do
        case T_Minus: do
            t := NEXT-TOKEN()
            e := PARSE-F()
            return MK-UNARY(Un_minus, e)
        case T_LParen: do
            t := NEXT-TOKEN()
            e := PARSE-E()
            if t == T_RParen then
                t := NEXT-TOKEN()
                return e
            else ERROR()
        case T_Ident: do
            e := MK-VAR(last-ident)
            t := NEXT-TOKEN()
            return e
        case T_Number: do
            e := MK-NUM(last-num)
            t := NEXT-TOKEN()
            return e
        otherwise do ERROR()
    end
end
\end{verbatim}
If a function ends with a recursive call of itself, as for example function \texttt{Parse-G()}, it is possible to replace this recursion with an iteration.

Functions \texttt{Parse-E()} and \texttt{Parse-G()} can be merged into one function.

Similarly, it is possible to replace a recursion with an iteration in function \texttt{Parse-U()}, and functions \texttt{Parse-T()} and \texttt{Parse-U()} can be merged into one function.
Parse-E ():
begin
  e1 := Parse-T()
  while t ∈ {T_Plus, T_Minus} do
    op := Parse-A()
    e2 := Parse-T()
    e1 := mk-binary(op, e1, e2)
  end
return e1
end

Parse-T ():
begin
  e1 := Parse-F()
  while t ∈ {T_Star, T_Slash} do
    op := Parse-M()
    e2 := Parse-F()
    e1 := mk-binary(op, e1, e2)
  end
return e1
end
Pushdown automata
Pushdown automaton

- We would like to accept language \( L = \{a^i b^i | i \geq 1\} \)
- We try to design a device (similar to finite automata) that reads a word and tells us if it belongs to language \( L \) or not.
- When reading \( a \)'s, we must remember the number of them to know how many \( b \)'s must follow.
We would like to accept language \( L = \{ a^i b^i \mid i \geq 1 \} \). We try to design a device (similar to finite automata) that reads a word and tells us if it belongs to language \( L \) or not.

When reading \( a \)'s, we must remember the number of them to know how many \( b \)'s must follow.

We can use a memory organized as a stack.

We write one symbol on the stack for each \( a \) that was read and remove one symbol from the stack for each \( b \).

If the stack is empty and all word has been read, the word belongs to the language.
Word $aaaabbb$ belongs to the language $L = \{a^i b^i \mid i \geq 1\}$
Word **aaaabbbb** belongs to the language \( L = \{ a^i b^i \mid i \geq 1 \} \)
Word \textit{aaaabbb} belongs to the language \( L = \{a^i b^i \mid i \geq 1\} \)
Word $aaaabbbb$ belongs to the language $L = \{a^i b^i \mid i \geq 1\}$
Word \(aaaabbbb\) belongs to the language \(L = \{a^i b^i \mid i \geq 1\}\)
Word \textit{aaaabbbb} belongs to the language $L = \{a^i b^i | i \geq 1\}$
Word $aaaabbbb$ belongs to the language $L = \{a^i b^i \mid i \geq 1\}$
Word $aaaabbbb$ belongs to the language $L = \{a^i b^i \mid i \geq 1\}$
Word \( aaaabbb \) belongs to the language \( L = \{ a^i b^i \mid i \geq 1 \} \)

The automaton has read all word and ends with the empty stack, and so the word is accepted by the automaton.
Word \textit{aaaabbb} does not belong to language \( L = \{a^i b^i \mid i \geq 1\} \)
Word *aaaabbb* does not belong to language $L = \{a^i b^i \mid i \geq 1\}$
Word $aaaabbb$ does not belong to language $L = \{a^i b^i \mid i \geq 1\}$
Word \textit{aaaabbb} does not belong to language \(L = \{a^i b^i \mid i \geq 1\}\).
Word $aaaabbb$ does not belong to language $L = \{a^i b^i \mid i \geq 1\}$
Word $aaaabbb$ does not belong to language $L = \{a^i b^i \mid i \geq 1\}$
Word \textit{aaaabbb} does not belong to language \[ L = \{a^i b^i \mid i \geq 1 \} \]
Word $aaaabbb$ does not belong to language $L = \{a^i b^i \mid i \geq 1\}$

The automaton has read all word but the stack is not empty and so the word is not accepted by the automaton.
Word $aaaabbbbb$ does not belong to language $L = \{a^i b^i \mid i \geq 1\}$
Word $aaaabbbbbb$ does not belong to language $L = \{a^i b^i \mid i \geq 1\}$
Word *aaaabbbbbb* does not belong to language $L = \{a^i b^i \mid i \geq 1\}$
Word \textit{aaaabbbbbb} does not belong to language $L = \{a^i b^i \mid i \geq 1\}$
Word $aaaabbbbb$ does not belong to language $L = \{a^i b^i \mid i \geq 1\}$.
Word $aaaabbbbbb$ does not belong to language $L = \{a^i b^i \mid i \geq 1\}$
Word \textit{aaaabbbbbb} does not belong to language \( L = \{a^i b^i | i \geq 1\} \)
Word \textit{aaaabbbbbb} does not belong to language \( L = \{a^i b^i \mid i \geq 1\} \)
Word $aaaabbbbbb$ does not belong to language $L = \{a^i b^i \mid i \geq 1\}$

The automaton reads $b$, it should remove a symbol from the stack but there is no symbol there. So the word is not accepted.

![Diagram showing the process of reading the word and the stack state](image-url)
Word $aababbab$ does not belong to language $L = \{a^i b^i \mid i \geq 1\}$
Word *aababbab* does not belong to language \( L = \{a^i b^i \mid i \geq 1\} \)
Pushdown automaton

- Word \textit{aababbab} does not belong to language \( L = \{a^i b^i \mid i \geq 1\} \)
Word $aababbab$ does not belong to language $L = \{a^i b^i \mid i \geq 1\}$
Word *aababbab* does not belong to language \( L = \{a^i b^i \mid i \geq 1\} \)

The automaton has read *a* but it is already in the state where it removes symbols from the stack, and so the word is not accepted.
A next step of the given pushdown automaton is always uniquely specified – the automaton is deterministic.

Is every context-free language accepted by some deterministic pushdown automaton?
A next step of the given pushdown automaton is always uniquely specified – the automaton is deterministic.

Is every context-free language accepted by some deterministic pushdown automaton?

Let us consider language \( L = \{ w(w)^R \mid w \in \{a, b\}^* \} \).

The first half of a word can be stored on the stack.

When reading the second part, the automaton removes the symbols from the stack if they are same as symbols in the input.

If the stack is empty after reading all word, the second is the same (the reverse of) the first.
Word $abaaba$ belongs to language $L = \{w(w)^R \mid w \in \{a, b\}^*\}$
Word *abaaba* belongs to language \( L = \{ w(w)^R \mid w \in \{a, b\}^* \} \)
Word *abaaba* belongs to language $L = \{ w(w)^R \mid w \in \{a, b\}^* \}$

When *a* is in the input, the control state is *S* and *B* is on the stack, the control state can be changed to *R* and *A* is pushed on the stack.
Word *abaaba* belongs to language $L = \{w(w)^R \mid w \in \{a, b\}^*\}$

When *a* is in the input, the control state is $S$ and $B$ is on the stack, the control state can be changed to $R$ and $A$ is pushed on the stack.
Word $abaaba$ belongs to language $L = \{w(w)^R \mid w \in \{a, b\}^*\}$

When $a$ is in the input, the control state is $S$ and $B$ is on the stack, the control state can be changed to $R$ and $A$ is pushed on the stack.
Word \textit{abaaba} belongs to language \( L = \{ w(w)^R \mid w \in \{a, b\}^* \} \)

When \( a \) is in the input, the control state is \( S \) and \( B \) is on the stack, the control state can be changed to \( R \) and \( A \) is pushed on the stack.
Word *abaaba* belongs to language $L = \{ w(w)^R \mid w \in \{a, b\}^* \}$

When *a* is in the input, the control state is *S* and *B* is on the stack, the control state can be changed to *R* and *A* is pushed on the stack.
Word \textit{abaaabaa} belongs to language $L = \{w(w)^R \mid w \in \{a, b\}^*\}$
Word *abaaababa* belongs to language $L = \{w(w)^R \mid w \in \{a,b\}^*\}$
Word *abaaba* belongs to language $L = \{w(w)^R \mid w \in \{a, b\}^*\}$
The word `abaaaba` belongs to the language $L = \{w(w)^R \mid w \in \{a, b\}^*\}$.
Word $abaaaaba$ belongs to language $L = \{w(w)^R \mid w \in \{a, b\}^*\}$
Word *abaaaaba* belongs to language $L = \{w(w)^R \mid w \in \{a, b\}^*\}$
Word \textit{abaaaaba} belongs to language $L = \{w(w)^R \mid w \in \{a, b\}^*\}$.
Word $abaaababa$ belongs to language $L = \{w(w)^R \mid w \in \{a, b\}^*\}$

[Diagram of a pushdown automaton with the word $abaaababa$ processed through it, showing the transition and stack evolution.]
Word *abaaaaba* belongs to language \( L = \{ w(w)^R \mid w \in \{a, b\}^* \} \)
The shown automaton nondeterministically “guesses” where is the half of the word.

It can be shown that there is no deterministic pushdown automaton accepting the given language.

Unlike finite automata, the deterministic version of pushdown automata are weaker than nondeterministic – they accept less languages.
A pushdown automaton (PDA) is a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ where

- $Q$ is a finite non-empty set of states
- $\Sigma$ is a finite non-empty set called an input alphabet
- $\Gamma$ is a finite non-empty set called a stack alphabet
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$ is a (nondeterministic) transition function
- $q_0 \in Q$ is the initial state
- $Z_0 \in \Gamma$ is the initial stack symbol
Let $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ be a pushdown automaton.

**Configurations of $\mathcal{M}$:**

- A *configuration* of a PDA is a triple $(q, w, \alpha)$ where $q \in Q$, $w \in \Sigma^*$, and $\alpha \in \Gamma^*$.

- A configuration $(q_0, w, Z_0)$, where $w \in \Sigma^*$, is called *initial*.
Steps performed by $M$:

- Binary relation $\vdash$ on configurations of $M$ represents the possible steps of computation performed by PDA $M$.

That $M$ can go from configuration $(q, w, \alpha)$ to configuration $(q', w', \alpha')$ is written as

$$(q, w, \alpha) \vdash (q', w', \alpha').$$

The relation $\vdash$ is defined as follows:

$$(q, aw, X\beta) \vdash (q', w, \alpha\beta) \iff (q', \alpha) \in \delta(q, a, X)$$

where $q, q' \in Q$, $a \in (\Sigma \cup \{\varepsilon\})$, $w \in \Sigma^*$, $X \in \Gamma$, and $\alpha, \beta \in \Gamma^*$. 
Computations of $\mathcal{M}$:

We define binary relation $\vdash^*$ on configurations of $\mathcal{M}$ as the reflexive and transitive closure of $\vdash$, i.e.,

$$(q, w, \alpha) \vdash^* (q', w', \alpha')$$

if there is a sequence of configurations

$$(q_0, w_0, \alpha_0), (q_1, w_1, \alpha_1), \ldots, (q_n, w_n, \alpha_n)$$

such that

1. $(q, w, \alpha) = (q_0, w_0, \alpha_0)$,
2. $(q', w', \alpha') = (q_n, w_n, \alpha_n)$, and
3. $(q_i, w_i, \alpha_i) \vdash (q_{i+1}, w_{i+1}, \alpha_{i+1})$ for each $i = 0, 1, \ldots, n - 1$, i.e.,

$$(q_0, w_0, \alpha_0) \vdash (q_1, w_1, \alpha_1) \vdash \ldots \vdash (q_n, w_n, \alpha_n)$$
Two different definitions acceptance of words are used:

- A pushdown automaton $\mathcal{M}$ accepting by an **empty stack** accepts a word $w$ iff there is some computation of $\mathcal{M}$ on $w$ such that $\mathcal{M}$ reads all symbols of $w$ and after reading them, the stack is empty.

- A pushdown automaton $\mathcal{M}$ accepting by an **accepting state** accepts a word $w$ iff there is some computation of $\mathcal{M}$ on $w$ such that $\mathcal{M}$ reads all symbols of $w$ and after reading them, the control unit of $\mathcal{M}$ is in some state from a given set of accepting states $F$. 
A word $w \in \Sigma^*$ is **accepted** by PDA $M$ by empty stack iff

$$(q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)$$

for some $q \in Q$.

**Definition**

The **language** $L(M)$ **accepted** by PDA $M$ by empty stack is defined as

$$L(M) = \{ w \in \Sigma^* | (\exists q \in Q)((q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)) \}.$$
Let us extend the definition of PDA $M$ with a set of **accepting states** $F$ (where $F \subseteq Q$).

- A word $w \in \Sigma^*$ is **accepted** by PDA $M$ by accepting state iff

  $$(q_0, w, Z_0) \vdash^* (q, \varepsilon, \alpha)$$

  for some $q \in F$ and $\alpha \in \Gamma^*$.

**Definition**

The **language** $L(M)$ accepted by PDA $M$ by accepting state is defined as

$$L(M) = \{ w \in \Sigma^* \mid (\exists q \in F)(\exists \alpha \in \Gamma^*)( (q_0, w, Z_0) \vdash^* (q, \varepsilon, \alpha) ) \}.$$
Example: $L = \{a^i b^i \mid i \geq 1\}$

$M = (Q, \Sigma, \Gamma, \delta, q_1, Z)$ where

- $Q = \{q_1, q_2\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{Z, I\}$

\[
\delta(q_1, a, Z) = \{(q_1, I)\} \quad \delta(q_1, b, Z) = \emptyset \\
\delta(q_1, a, I) = \{(q_1, II)\} \quad \delta(q_1, b, I) = \{(q_2, \varepsilon)\} \\
\delta(q_2, a, I) = \emptyset \quad \delta(q_2, b, I) = \{(q_2, \varepsilon)\} \\
\delta(q_2, a, Z) = \emptyset \quad \delta(q_2, b, Z) = \emptyset
\]

Remark: We often omit those values of transition function $\delta$ that are $\emptyset$. 
**Example:** \( L = \{ w(w)^R \mid w \in \{ a, b \}^* \} \)

\( M = ( Q, \Sigma, \Gamma, \delta, U, Z ) \) where

- \( Q = \{ q_1, q_2 \} \)
- \( \Sigma = \{ a, b \} \)
- \( \Gamma = \{ Z, A, B \} \)

\[
\begin{align*}
\delta(q_1, a, Z) &= \{ (q_1, A) \} \\
\delta(q_1, a, A) &= \{ (q_1, AA), (q_2, AA) \} \\
\delta(q_1, a, B) &= \{ (q_1, AB), (q_2, AB) \} \\
\delta(q_1, \varepsilon, Z) &= \{ (q_1, \varepsilon) \} \\
\delta(q_1, \varepsilon, A) &= \emptyset \\
\delta(q_1, \varepsilon, B) &= \emptyset \\
\delta(q_2, a, Z) &= \emptyset \\
\delta(q_2, a, A) &= \{ (q_2, \varepsilon) \} \\
\delta(q_2, a, B) &= \emptyset \\
\delta(q_2, \varepsilon, Z) &= \emptyset \\
\delta(q_2, \varepsilon, A) &= \emptyset \\
\delta(q_2, \varepsilon, B) &= \emptyset \\
\delta(q_1, b, Z) &= \{ (q_1, B) \} \\
\delta(q_1, b, A) &= \{ (q_1, BA), (q_2, BA) \} \\
\delta(q_1, b, B) &= \{ (q_1, BB), (q_2, BB) \} \\
\delta(q_2, b, A) &= \emptyset \\
\delta(q_2, b, B) &= \{ (q_2, \varepsilon) \} \\
\delta(q_2, \varepsilon, Z) &= \emptyset \\
\delta(q_2, \varepsilon, A) &= \emptyset \\
\delta(q_2, \varepsilon, B) &= \emptyset 
\end{align*}
\]
Lemma

For every context-free grammar $G$ we can construct a pushdown automaton $M$ (with one control state) such that $L(M) = L(G)$.

Proof: For CFG $G = (\Pi, \Sigma, S, P)$ we construct PDA $M = (\{q_0\}, \Sigma, \Pi \cup \Sigma, \delta, q_0, S)$, where

- for each $X \in \Pi$: $\delta(q_0, \varepsilon, X) = \{(q_0, \alpha) \mid (X \rightarrow \alpha) \in P\}$,
- for each $a \in \Sigma$: $\delta(q_0, a, a) = \{(q_0, \varepsilon)\}$,
- the value of $\delta$ is $\emptyset$ in all other cases.

It can be shown by induction that

$S \Rightarrow^* u\alpha$ (in $G$) \iff $(q_0, u, S) \vdash^* (q_0, \varepsilon, \alpha)$ (in $M$)

where $u \in \Sigma^*$ and $\alpha \in (\{\varepsilon\} \cup \Pi(\Pi \cup \Sigma)^*)$. 
Example: CFG $G = ({S}, \{a, b\}, S, P)$ where $P$ contains rules

$$S \rightarrow aSb \mid ab$$

(and so $L(G) = \{a^i b^i \mid i \geq 1\}$)
Equivalence of CFG and PDA

Example: CFG $G = (\{S\}, \{a, b\}, S, P)$ where $P$ contains rules

$$S \rightarrow aSb \mid ab$$

(and so $L(G) = \{a^i b^i \mid i \geq 1\}$)

We construct PDA $M = (\{q_0\}, \{a, b\}, \{S, a, b\}, \delta, q_0, S)$, where

$\delta(q_0, \varepsilon, S) = \{(q_0, aSb), (q_0, ab)\}$

$\delta(q_0, a, a) = \{(q_0, \varepsilon)\}$

$\delta(q_0, b, b) = \{(q_0, \varepsilon)\}$
Equivalence of CFG and PDA

Example: CFG $G = (\{S\}, \{a, b\}, S, P)$ where $P$ contains rules

$$S \rightarrow aSb \mid ab$$

(and so $L(G) = \{a^i b^i \mid i \geq 1\}$)

We construct PDA $M = (\{q_0\}, \{a, b\}, \{S, a, b\}, \delta, q_0, S)$, where

$\delta(q_0, \varepsilon, S) = \{(q_0, aSb), (q_0, ab)\}$

$\delta(q_0, a, a) = \{(q_0, \varepsilon)\}$

$\delta(q_0, b, b) = \{(q_0, \varepsilon)\}$

Example of derivation of $aaaabbbb$ in $G$ and the corresponding computation of $M$: $S \Rightarrow aSb$

$(q_0, aaaabbbb, S) \vdash (q_0, aaaabbbb, aSb)$
Equivalence of CFG and PDA

Example: CFG $G = (\{S\}, \{a, b\}, S, P)$ where $P$ contains rules

\[
S \rightarrow aSb \mid ab
\]

(and so $L(G) = \{a^i b^i \mid i \geq 1\}$)

We construct PDA $M = (\{q_0\}, \{a, b\}, \{S, a, b\}, \delta, q_0, S)$, where

\[
\delta(q_0, \epsilon, S) = \{(q_0, aSb), (q_0, ab)\}
\]
\[
\delta(q_0, a, a) = \{(q_0, \epsilon)\}
\]
\[
\delta(q_0, b, b) = \{(q_0, \epsilon)\}
\]

Example of derivation of $aaaabbbb$ in $G$ and the corresponding computation of $M$: $S \Rightarrow aSb$

$(q_0, aaaabbbb, S) \vdash (q_0, aaaabbbb, aSb) \vdash (q_0, aaabb, Sb)$
Equivalence of CFG and PDA

Example: CFG $G = (\{S\}, \{a, b\}, S, P)$ where $P$ contains rules

$$S \rightarrow aSb \mid ab$$

(and so $L(G) = \{a^i b^i \mid i \geq 1\}$)

We construct PDA $M = (\{q_0\}, \{a, b\}, \{S, a, b\}, \delta, q_0, S)$, where

$\delta(q_0, \varepsilon, S) = \{(q_0, aSb), (q_0, ab)\}$
$\delta(q_0, a, a) = \{(q_0, \varepsilon)\}$
$\delta(q_0, b, b) = \{(q_0, \varepsilon)\}$

Example of derivation of $aaaabbbb$ in $G$ and the corresponding computation of $M$: $S \Rightarrow aSb \Rightarrow aaSbb$

$(q_0, aaaabbbb, S) \vdash (q_0, aaaabbbb, aSb) \vdash (q_0, aabbb, Sb) \vdash (q_0, aabbb, aSbb)$
Equivalence of CFG and PDA

Example: CFG \( G = (\{S\}, \{a, b\}, S, P) \) where \( P \) contains rules

\[
S \rightarrow aSb | ab
\]

(and so \( L(G) = \{a^i b^i \mid i \geq 1\} \))

We construct PDA \( M = (\{q_0\}, \{a, b\}, \{S, a, b\}, \delta, q_0, S) \), where

\[
\begin{align*}
\delta(q_0, \varepsilon, S) &= \{(q_0, aSb), (q_0, ab)\} \\
\delta(q_0, a, a) &= \{(q_0, \varepsilon)\} \\
\delta(q_0, b, b) &= \{(q_0, \varepsilon)\}
\end{align*}
\]

Example of derivation of \( aaaabbbb \) in \( G \) and the corresponding computation of \( M \): \( S \Rightarrow aSb \Rightarrow aaSbb \)

\[
(q_0, aaaabbbb, S) \vdash (q_0, aaaabbbb, aSb) \vdash (q_0, aaabbb, Sb) \\
\vdash (q_0, aaabbb, aSbb) \vdash (q_0, aabbbb, Sbb)
\]
Equivalence of CFG and PDA

Example: CFG $G = (\{S\}, \{a, b\}, S, P)$ where $P$ contains rules

\[ S \to aSb \mid ab \]

(and so $L(G) = \{a^i b^i \mid i \geq 1\}$)

We construct PDA $M = (\{q_0\}, \{a, b\}, \{S, a, b\}, \delta, q_0, S)$, where

\[
\begin{align*}
\delta(q_0, \varepsilon, S) &= \{ (q_0, aSb), (q_0, ab) \} \\
\delta(q_0, a, a) &= \{ (q_0, \varepsilon) \} \\
\delta(q_0, b, b) &= \{ (q_0, \varepsilon) \}
\end{align*}
\]

Example of derivation of $aaaabbbb$ in $G$ and the corresponding computation of $M$: $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb$

\[
\begin{align*}
(q_0, aaaabbbb, S) &\vdash (q_0, aaaabbbb, aSb) \vdash (q_0, aaabbbb, Sb) \\
&\vdash (q_0, aaabbbb, aSbb) \vdash (q_0, aabbbb, Sbb) \vdash (q_0, aabbbb, aSbbb)
\end{align*}
\]
Equivalence of CFG and PDA

Example: CFG $G = (\{S\}, \{a, b\}, S, P)$ where $P$ contains rules

$$S \to aSb | ab$$

(and so $L(G) = \{a^i b^i \mid i \geq 1\}$)

We construct PDA $M = (\{q_0\}, \{a, b\}, \{S, a, b\}, \delta, q_0, S)$, where

\[
\begin{align*}
\delta(q_0, \epsilon, S) &= \{(q_0, aSb), (q_0, ab)\} \\
\delta(q_0, a, a) &= \{(q_0, \epsilon)\} \\
\delta(q_0, b, b) &= \{(q_0, \epsilon)\}
\end{align*}
\]

Example of derivation of $aaaabbbb$ in $G$ and the corresponding computation of $M$: $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb$

\[
\begin{align*}
(q_0, aaaabbbb, S) \vdash (q_0, aaaabbbb, aSb) \vdash (q_0, aaabbbb, Sb) \\
\vdash (q_0, aaabbbb, aSbb) \vdash (q_0, aabbbb, Sbb) \vdash (q_0, aabbbb, aSbbb) \\
\vdash (q_0, aabbbb, Sbbb)
\end{align*}
\]
Equivalence of CFG and PDA

Example: CFG $G = (\{S\}, \{a, b\}, S, P)$ where $P$ contains rules

\[ S \rightarrow aSb \mid ab \]

(and so $L(G) = \{a^i b^i \mid i \geq 1\}$)

We construct PDA $M = (\{q_0\}, \{a, b\}, \{S, a, b\}, \delta, q_0, S)$, where

\[
\begin{align*}
\delta(q_0, \varepsilon, S) &= \{(q_0, aSb), (q_0, ab)\} \\
\delta(q_0, a, a) &= \{(q_0, \varepsilon)\} \\
\delta(q_0, b, b) &= \{(q_0, \varepsilon)\}
\end{align*}
\]

Example of derivation of $aaaabbbb$ in $G$ and the corresponding computation of $M$:

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaabbbb$

\[
\begin{align*}
(q_0, aaaabbbb, S) &\vdash (q_0, aaaabbbb, aSb) &\vdash (q_0, aaabbbb, Sb) \\
\vdash (q_0, aaabbbb, aSbb) &\vdash (q_0, aabbbb, Sbb) &\vdash (q_0, aabbbb, aSbbb) \\
\vdash (q_0, abbbb, Sbbb) &\vdash (q_0, abbbb, abbbb)
\end{align*}
\]

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019  234 / 490
Equivalence of CFG and PDA

Example: CFG $G = (\{S\}, \{a, b\}, S, P)$ where $P$ contains rules

\[ S \rightarrow aSb | ab \]

(and so $L(G) = \{a^i b^i | i \geq 1\}$)

We construct PDA $M = (\{q_0\}, \{a, b\}, \{S, a, b\}, \delta, q_0, S)$, where

\[
\begin{align*}
\delta(q_0, \varepsilon, S) &= \{(q_0, aSb), (q_0, ab)\} \\
\delta(q_0, a, a) &= \{(q_0, \varepsilon)\} \\
\delta(q_0, b, b) &= \{(q_0, \varepsilon)\}
\end{align*}
\]

Example of derivation of $aaaabbbb$ in $G$ and the corresponding computation of $M$:

\[ S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaaabbbb \]

\[
\begin{align*}
(q_0, aaaaabbbb, S) \vdash (q_0, aaaaabbbb, aSb) \vdash (q_0, aaabbbb, Sb) \\
\vdash (q_0, aaabbbb, aSbb) \vdash (q_0, aabbbb, Sbb) \vdash (q_0, aabbbb, aSbbb) \\
\vdash (q_0, abbbb, Sbb) \vdash (q_0, abbbb, abbbb) \vdash (q_0, bbbb, bbbb)
\end{align*}
\]
Equivalence of CFG and PDA

Example: CFG $G = (\{S\}, \{a, b\}, S, P)$ where $P$ contains rules

$$S \rightarrow aSb \mid ab$$

(and so $L(G) = \{a^i b^i \mid i \geq 1\}$)

We construct PDA $M = (\{q_0\}, \{a, b\}, \{S, a, b\}, \delta, q_0, S)$, where

$\delta(q_0, \varepsilon, S) = \{(q_0, aSb), (q_0, ab)\}$

$\delta(q_0, a, a) = \{(q_0, \varepsilon)\}$

$\delta(q_0, b, b) = \{(q_0, \varepsilon)\}$

Example of derivation of $aaaabbbb$ in $G$ and the corresponding computation of $M$: $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaaabbbb$

$(q_0, aaaaabbbb, S) \vdash (q_0, aaaaabbbb, aSb) \vdash (q_0, aabbbb, Sb)$

$\vdash (q_0, aabbbb, aSbb) \vdash (q_0, addbbb, Sbb) \vdash (q_0, addbb, aSbbb)$

$\vdash (q_0, bbbb, Sbbb) \vdash (q_0, bbbb, abbbb) \vdash (q_0, bbbb, bbbb)$

$\vdash (q_0, bbb, bbb) \vdash (q_0, bb, bb) \vdash (q_0, b, b) \vdash (q_0, \varepsilon, \varepsilon)$
Equivalent of CFG and PDA

Lemma

For every pushdown automaton \( \mathcal{M} \) with one control state, there is a corresponding CFG \( \mathcal{G} \) such \( L(\mathcal{G}) = L(\mathcal{M}) \).

Proof: For PDA \( \mathcal{M} = (\{q_0\}, \Sigma, \Gamma, \delta, q_0, Z_0) \), where \( \Sigma \cap \Gamma = \emptyset \), we construct CFG \( \mathcal{G} = (\Gamma, \Sigma, Z_0, P) \), where

\[
(A \rightarrow a\alpha) \in P \quad \text{iff} \quad (q_0, \alpha) \in \delta(q_0, a, A)
\]

for all \( A \in \Gamma, \ a \in \Sigma \cup \{\varepsilon\}, \) and \( \alpha \in \Gamma^* \).

It can be proved by induction that

\[
Z_0 \Rightarrow^* u\alpha \quad \text{(in} \ \mathcal{M}) \quad \text{iff} \quad (q_0, u, Z_0) \vdash^*_\mathcal{M} (q_0, \varepsilon, \alpha) \quad \text{(in} \ \mathcal{G})
\]

where \( u \in \Sigma^* \) and \( \alpha \in \Gamma^* \).
Lemma

For every pushdown automaton $M$ there exists a pushdown automaton $M'$ with one control state such that $L(M') = L(M)$.

Proof idea:

- The control state of $M$ is stored on the top of the stack of $M'$.
- For $\delta(q, a, X) = \{(q', \varepsilon)\}$ we must ensure that the new control state on the stack of $M'$ is $q'$. (Other cases are straightforward.)
- Stack symbols of $M'$ are triples of the form $(q, A, q')$ where $q$ represents the control state of $M$ when that symbol is on the top, $A$ is the stack symbol of $M$, and $q'$ is the first control state in the triple below it.
- PDA $M'$ nondeterministically “guesses” the control states to which $M$ goes when the given stack symbols becomes the top of the stack.
Proposition

For every context-free grammar $\mathcal{G}$ there is some (nondeterministic) pushdown automaton $\mathcal{M}$ such that $L(\mathcal{G}) = L(\mathcal{M})$.

Proposition

For every pushdown automaton $\mathcal{M}$ there is some context-free grammar $\mathcal{G}$ such that $L(\mathcal{M}) = L(\mathcal{G})$. 
Limitations of Context-free Languages

- Context-free grammars and pushdown automata are not able to generate or accept all possible languages.
- Some examples of languages for which it can be proved that they are not context-free (i.e., that there is no context-free grammar generating them):
  
  $$L_1 = \{a^i b^i c^i \mid i \geq 0\}$$
  $$L_2 = \{ww \mid w \in \{a, b\}^*\}$$
Lemma

If a language $L$ is context-free then there is $n \in \mathbb{N}$ such that for all $z \in L$, such that $|z| \geq n$, there are words $u, v, w, x, y \in \Sigma^*$ such that $z = uvwxy$, $|vx| \geq 1$, $|vwx| \leq n$, and for all $i \in \mathbb{N}$ it holds that $uv^iwx^iy \in L$. 
Pumping Lemma for Context-free Languages

Lemma

If a language $L$ is context-free then there is $n \in \mathbb{N}$ such that for all $z \in L$, such that $|z| \geq n$, there are words $u, v, w, x, y \in \Sigma^*$ such that $z = uvwxy$, $|vx| \geq 1$, $|vwx| \leq n$, and for all $i \in \mathbb{N}$ it holds that $uv^iwx^iy \in L$.

Again, this can be formulated as a game:

1. Player I chooses $n \in \mathbb{N}$.
2. Player II chooses a word $z \in L$ such that $|z| \geq n$.
3. Player I chooses words $u, v, w, x, y \in \Sigma^*$ such that $z = uvwxy$, $|vx| \geq 1$, and $|vwx| \leq n$.
4. Player II chooses $i \in \mathbb{N}$.
5. If $uv^iwx^iy \in L$ then Player I wins, otherwise Player II wins.

If $L$ is context-free then Player I has a winning strategy. (So if Player II has a winning strategy, then $L$ is not context-free.)
We extend a deterministic finite automaton with the possibility of moving the head in both directions, of writing symbols on the tape, and we extend its tape into infinity.
Turing Machine

Definition

Formally, **Turing machine** is a tuple $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

- $Q$ is a finite non-empty set of *states*
- $\Gamma$ is a finite (non-empty) set of *tape symbols* (tape alphabet)
- $\Sigma \subseteq \Gamma$ is a finite non-empty set of *input symbols* (input alphabet)
- $\delta : (Q - F) \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 0, +1\}$ is a *transition function*
- $q_0 \in Q$ is an *initial state*
- $F \subseteq Q$ is a set of *final states*
We assume that $\Gamma - \Sigma$ always contains a special element $\square$ denoting a blank symbol.

A configuration is given by:
- a state of the control unit
- a content of the tape
- a position of the head

A computation over a word $w \in \Sigma^*$ starts in the initial configuration where:
- the state of the control unit is $q_0$
- word $w$ is written on the tape, remaining cells of the tape are filled with the blank symbols ($\square$)
- the head is on the first symbol of the word $w$ (or on symbol $\square$ when $w = \varepsilon$)
One step of a Turing machine:

Let us assume that:

- the state of the control unit is $q$
- the cell of the tape on the position of the head contains symbol $b$

Let us say that $\delta(q, b) = (q', b', d)$ where $d \in \{-1, 0, 1\}$.

One step of the Turing machine is performed as follows:

- the state of the control unit is changed to $q'$
- symbol $b'$ is written on the tape cell on the position of the head instead of $b$
- The head is moved depending on $d$:
  - for $d = -1$ the head is moved one cell left
  - for $d = 1$ the head is moved one cell right
  - for $d = 0$ the position of the head is not changed
If the state of the control unit belongs to the set $F$ then the next step is not defined and the computation halts.

We often choose the set of final states $F = \{q_{\text{yes}}, q_{\text{no}}\}$.

Then we can define for a word $w \in \Sigma^*$ if a given Turing machine accepts it:

- If the state of the control unit after the computation over the word $w$ is $q_{\text{yes}}$, the machine accepts the word $w$.
- If the state of the control unit after the computation over the word $w$ is $q_{\text{no}}$, the machine does not accept the word $w$.
- The computation of the machine over the word $w$ can be infinite. In this case the machine does not accept the word $w$.

The language $L(M)$ of a Turing machine $M$ is the set of all words accepted by $M$. 
A Turing machine can give not only an answer \textbf{Yes} or \textbf{No} but it can also compute a function that assigns to each word from \( \Sigma^* \) some other word (from \( \Gamma^* \)).

A word assigned to a word \( w \) is the word that remains on the tape after the computation over the word \( w \) when we remove all symbols \( \Box \).
### Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[
Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\}
\]

\[
\Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\}
\]

<table>
<thead>
<tr>
<th>State</th>
<th>(\square)</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>(q_{\text{yes}}, \square, 0)</td>
<td>(q_1, x, +1)</td>
<td>(q_{\text{no}}, b, 0)</td>
<td>(q_{\text{no}}, c, 0)</td>
<td>(q_0, x, +1)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>(q_{\text{no}}, \square, 0)</td>
<td>(q_1, a, +1)</td>
<td>(q_2, x, +1)</td>
<td>(q_{\text{no}}, c, 0)</td>
<td>(q_1, x, +1)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>(q_{\text{no}}, \square, 0)</td>
<td>(q_{\text{no}}, a, 0)</td>
<td>(q_2, b, +1)</td>
<td>(q_3, x, +1)</td>
<td>(q_2, x, +1)</td>
</tr>
<tr>
<td>(q_3)</td>
<td>(q_4, \square, -1)</td>
<td>(q_{\text{no}}, a, 0)</td>
<td>(q_{\text{no}}, b, 0)</td>
<td>(q_3, c, +1)</td>
<td>(q_3, x, +1)</td>
</tr>
<tr>
<td>(q_4)</td>
<td>(q_0, \square, +1)</td>
<td>(q_4, a, -1)</td>
<td>(q_4, b, -1)</td>
<td>(q_4, c, -1)</td>
<td>(q_4, x, -1)</td>
</tr>
</tbody>
</table>

- **Diagram:**
  - Initial state: \(q_0\)
  - Transition: \(q_0\) to \(q_0\) with input \(\square\), \(a\), \(b\), \(c\), \(x\)
  - Transition: \(q_0\) to \(q_1\) with input \(a\), \(b\), \(c\), \(x\)
  - Transition: \(q_0\) to \(q_2\) with input \(a\), \(b\), \(c\), \(x\)
  - Transition: \(q_0\) to \(q_3\) with input \(a\), \(b\), \(c\), \(x\)
  - Transition: \(q_0\) to \(q_4\) with input \(a\), \(b\), \(c\), \(x\)

- **State Transition:**
  - \(q_0\) to \(q_1\) with input \(a\), \(b\), \(c\), \(x\)
  - \(q_0\) to \(q_2\) with input \(a\), \(b\), \(c\), \(x\)
  - \(q_0\) to \(q_3\) with input \(a\), \(b\), \(c\), \(x\)
  - \(q_0\) to \(q_4\) with input \(a\), \(b\), \(c\), \(x\)
Turing Machine

Language \( L = \{a^n b^n c^n | n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_{\text{no}}, \square, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_{\text{no}}, \square, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>

\[ q_1 \]

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019 246 / 490
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[ \delta \]

\[ \begin{array}{|c|c|c|c|c|}
\hline
q & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array} \]
Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\Box, a, b, c, x\} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \Box )</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \Box, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \Box, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \Box, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_{\text{no}}, \Box, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \Box, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\[ \Sigma = \{ a, b, c \} \quad \Gamma = \{ \sqcup, a, b, c, x \} \]

\[
\begin{array}{c|cccccc}
\delta & \sqcup & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \sqcup, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \sqcup, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \sqcup, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \sqcup, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \sqcup, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
**Turing Machine**

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \]

\[ F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \]

\[ \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4, \square, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \square, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  \quad $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$  \quad $\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>
**Turing Machine**

Language \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\[ \Sigma = \{ a, b, c \} \quad \Gamma = \{ \Box, a, b, c, x \} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \Box )</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \Box, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \Box, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \Box, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_{\text{no}}, \Box, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_{\text{no}}, \Box, +1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, -1 )</td>
<td>( q_0, x, -1 )</td>
</tr>
</tbody>
</table>

**Diagram:**

- States: \( q_2 \)
- Input: \( a, a, a, x, b, b, b, c, c, c, c \)
- Tape: \( \Box, \Box \)
Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\}$  
$F = \{q_{yes}, q_{no}\}$

$\Sigma = \{a, b, c\}$  
$\Gamma = \{\square, a, b, c, x\}$
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

| State | \(\square\) | \(a\) & \(b\) & \(c\) & \(x\) |
|-------|-------------|-------------|-------------|-------------|
| \(q_0\) | \(q_{\text{yes}}, \square, 0\) | \(q_1, x, +1\) | \(q_{\text{no}}, b, 0\) | \(q_{\text{no}}, c, 0\) | \(q_0, x, +1\) |
| \(q_1\) | \(q_{\text{no}}, \square, 0\) | \(q_1, a, +1\) | \(q_2, x, +1\) | \(q_{\text{no}}, c, 0\) | \(q_1, x, +1\) |
| \(q_2\) | \(q_{\text{no}}, \square, 0\) | \(q_{\text{no}}, a, 0\) | \(q_2, b, +1\) | \(q_3, x, +1\) | \(q_2, x, +1\) |
| \(q_3\) | \(q_4, \square, -1\) | \(q_{\text{no}}, a, 0\) | \(q_{\text{no}}, b, 0\) | \(q_3, c, +1\) | \(q_3, x, +1\) |
| \(q_4\) | \(q_0, \square, +1\) | \(q_4, a, -1\) | \(q_4, b, -1\) | \(q_4, c, -1\) | \(q_4, x, -1\) |
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( q )</th>
<th>( \square )</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( (q_{\text{yes}}, \square, 0) )</td>
<td>( (q_1, x, +1) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_0, x, +1) )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_1, a, +1) )</td>
<td>( (q_2, x, +1) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_1, x, +1) )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_2, b, +1) )</td>
<td>( (q_3, x, +1) )</td>
<td>( (q_2, x, +1) )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( (q_4, \square, -1) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_3, c, +1) )</td>
<td>( (q_3, x, +1) )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( (q_0, \square, +1) )</td>
<td>( (q_4, a, -1) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_4, c, -1) )</td>
<td>( (q_{\text{no}}, x, -1) )</td>
</tr>
</tbody>
</table>
**Turing Machine**

Language \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[
Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \}
\]

\( \Sigma = \{ a, b, c \} \quad \Gamma = \{ \square, a, b, c, x \} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( (q_{\text{yes}}, \square, 0) )</td>
<td>( (q_1, x, +1) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_0, x, +1) )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_1, a, +1) )</td>
<td>( (q_2, x, +1) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_1, x, +1) )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_2, b, +1) )</td>
<td>( (q_3, x, +1) )</td>
<td>( (q_2, x, +1) )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( (q_4, \square, -1) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_3, c, +1) )</td>
<td>( (q_3, x, +1) )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( (q_0, \square, +1) )</td>
<td>( (q_4, a, -1) )</td>
<td>( (q_4, b, -1) )</td>
<td>( (q_4, c, -1) )</td>
<td>( (q_4, x, -1) )</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4, \square, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \square, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

\[
\begin{array}{cccccc}
\delta & \square & a & b & c & x \\
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, a, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, b, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \)

\( F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \)

\( \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ \quad $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ \quad $\Gamma = \{\Box, a, b, c, x\}$

\[
\begin{array}{c|ccccc}
\delta & \Box & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \Box, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \Box, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \Box, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \Box, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \Box, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]

$q_4$
Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\} \quad F = \{q_{yes}, q_{no}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\Box, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>(\delta)</th>
<th>(\Box)</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>(q_{yes}, \Box, 0)</td>
<td>(q_1, x, +1)</td>
<td>(q_{no}, b, 0)</td>
<td>(q_{no}, c, 0)</td>
<td>(q_0, x, +1)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>(q_{no}, \Box, 0)</td>
<td>(q_1, a, +1)</td>
<td>(q_2, x, +1)</td>
<td>(q_{no}, c, 0)</td>
<td>(q_1, x, +1)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>(q_{no}, \Box, 0)</td>
<td>(q_{no}, a, 0)</td>
<td>(q_{no}, b, +1)</td>
<td>(q_{no}, c, 0)</td>
<td>(q_2, x, +1)</td>
</tr>
<tr>
<td>(q_3)</td>
<td>(q_{no}, \Box, -1)</td>
<td>(q_{no}, a, 0)</td>
<td>(q_{no}, b, 0)</td>
<td>(q_{no}, c, +1)</td>
<td>(q_3, x, +1)</td>
</tr>
<tr>
<td>(q_4)</td>
<td>(q_0, \Box, +1)</td>
<td>(q_4, a, -1)</td>
<td>(q_4, b, -1)</td>
<td>(q_4, c, -1)</td>
<td>(q_4, x, -1)</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\} \)  \( F = \{q_{yes}, q_{no}\} \)

\( \Sigma = \{a, b, c\} \)  \( \Gamma = \{\square, a, b, c, x\} \)

\[ \begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{yes}, \square, 0) & (q_1, x, +1) & (q_{no}, b, 0) & (q_{no}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{no}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{no}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{no}, \square, 0) & (q_{no}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{no}, a, 0) & (q_{no}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_{no}, a, 0) & (q_{no}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array} \]
### Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad \text{and} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4, \square, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \square, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>

---

![Diagram](attachment:diagram.png)
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ $\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>

$q_4$
Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  \quad  $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$  \quad  $\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|ccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
### Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>State</th>
<th>(\square)</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>(q_{\text{yes}}, \square, 0)</td>
<td>(q_1, x, +1)</td>
<td>(q_{\text{no}}, b, 0)</td>
<td>(q_{\text{no}}, c, 0)</td>
<td>(q_0, x, +1)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>(q_{\text{no}}, \square, 0)</td>
<td>(q_1, a, +1)</td>
<td>(q_2, x, +1)</td>
<td>(q_{\text{no}}, c, 0)</td>
<td>(q_1, x, +1)</td>
</tr>
<tr>
<td>(q_2)</td>
<td>(q_{\text{no}}, \square, 0)</td>
<td>(q_{\text{no}}, a, 0)</td>
<td>(q_2, b, +1)</td>
<td>(q_3, x, +1)</td>
<td>(q_2, x, +1)</td>
</tr>
<tr>
<td>(q_3)</td>
<td>(q_{4}, \square, -1)</td>
<td>(q_{\text{no}}, a, 0)</td>
<td>(q_{\text{no}}, b, 0)</td>
<td>(q_3, c, +1)</td>
<td>(q_3, x, +1)</td>
</tr>
<tr>
<td>(q_4)</td>
<td>(q_0, \square, +1)</td>
<td>(q_{4}, a, -1)</td>
<td>(q_{4}, b, -1)</td>
<td>(q_{4}, c, -1)</td>
<td>(q_4, x, -1)</td>
</tr>
</tbody>
</table>
**Turing Machine**

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \)

\( F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \)

\( \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\} \quad F = \{q_{yes}, q_{no}\} \]

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( (q_{yes}, \square, 0) )</td>
<td>( (q_1, x, +1) )</td>
<td>( (q_{no}, b, 0) )</td>
<td>( (q_{no}, c, 0) )</td>
<td>( (q_{0}, x, +1) )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_{no}, \square, 0) )</td>
<td>( (q_1, a, +1) )</td>
<td>( (q_2, x, +1) )</td>
<td>( (q_{no}, c, 0) )</td>
<td>( (q_{1}, x, +1) )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( (q_{no}, \square, 0) )</td>
<td>( (q_{no}, a, 0) )</td>
<td>( (q_2, b, +1) )</td>
<td>( (q_3, x, +1) )</td>
<td>( (q_{2}, x, +1) )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( (q_4, \square, -1) )</td>
<td>( (q_{no}, a, 0) )</td>
<td>( (q_{no}, b, 0) )</td>
<td>( (q_3, c, +1) )</td>
<td>( (q_{3}, x, +1) )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( (q_0, \square, +1) )</td>
<td>( (q_4, a, -1) )</td>
<td>( (q_4, b, -1) )</td>
<td>( (q_4, c, -1) )</td>
<td>( (q_{4}, x, -1) )</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\( Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \)

\( \Sigma = \{ a, b, c \} \quad \Gamma = \{ \square, a, b, c, x \} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>(( q_{\text{yes}}, \square, 0 ))</td>
<td>(( q_1, x, +1 ))</td>
<td>(( q_{\text{no}}, b, 0 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
<td>(( q_0, x, +1 ))</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>(( q_{\text{no}}, \square, 0 ))</td>
<td>(( q_1, a, +1 ))</td>
<td>(( q_2, x, +1 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
<td>(( q_1, x, +1 ))</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>(( q_{\text{no}}, \square, 0 ))</td>
<td>(( q_{\text{no}}, a, 0 ))</td>
<td>(( q_2, b, +1 ))</td>
<td>(( q_3, x, +1 ))</td>
<td>(( q_2, x, +1 ))</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>(( q_{\text{no}}, \square, -1 ))</td>
<td>(( q_{\text{no}}, a, 0 ))</td>
<td>(( q_{\text{no}}, b, 0 ))</td>
<td>(( q_3, c, +1 ))</td>
<td>(( q_3, x, +1 ))</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>(( q_0, \square, +1 ))</td>
<td>(( q_4, a, -1 ))</td>
<td>(( q_4, b, -1 ))</td>
<td>(( q_4, c, -1 ))</td>
<td>(( q_4, x, -1 ))</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{|c|c|c|c|c|}
\hline
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

\[
\begin{array}{c|ccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]

\[
\begin{array}{cccccccccccccc}
\square & \square & x & x & a & a & x & b & b & b & x & c & c & c & \square & \square & \square \\
\end{array}
\]

\( q_1 \)
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ \quad $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ \quad $\Gamma = \{\square, a, b, c, x\}$

$\delta$

<table>
<thead>
<tr>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, 1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, 1)$</td>
<td>$(q_2, x, 1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, 1)$</td>
<td>$(q_3, x, 1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, 1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, 1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
</tr>
</tbody>
</table>
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$  $\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>

$q_2$
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\}$  \hspace{1cm} $F = \{q_{yes}, q_{no}\}$

$\Sigma = \{a, b, c\}$ \hspace{1cm} $\Gamma = \{\alla, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\alla$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{yes}, \alla, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{no}, \alla, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{no}, \alla, 0)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \alla, -1)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \alla, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>

$q_2$
**Turing Machine**

**Language** \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \]

\[ F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\[ \Sigma = \{ a, b, c \} \]

\[ \Gamma = \{ \square, a, b, c, x \} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4, \square, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \square, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>

Diagram:

```
□ □ x x a a x x b b x c c c □ □ □
```

\( q_2 \)
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ \hspace{1cm} $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ \hspace{1cm} $\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>

$q_2$
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\Box, a, b, c, x\} \]

\[ \delta \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \Box )</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>(q_{\text{yes}}, \Box, 0)</td>
<td>(q_1, x, +1)</td>
<td>(q_{\text{no}}, b, 0)</td>
<td>(q_{\text{no}}, c, 0)</td>
<td>(q_0, x, +1)</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>(q_{\text{no}}, \Box, 0)</td>
<td>(q_1, a, +1)</td>
<td>(q_2, x, +1)</td>
<td>(q_{\text{no}}, c, 0)</td>
<td>(q_1, x, +1)</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>(q_{\text{no}}, \Box, 0)</td>
<td>(q_{\text{no}}, a, 0)</td>
<td>(q_2, b, +1)</td>
<td>(q_3, x, +1)</td>
<td>(q_2, x, +1)</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>(q_4, \Box, -1)</td>
<td>(q_{\text{no}}, a, 0)</td>
<td>(q_{\text{no}}, b, 0)</td>
<td>(q_3, c, +1)</td>
<td>(q_3, x, +1)</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>(q_0, \Box, +1)</td>
<td>(q_{\text{no}}, a, 0)</td>
<td>(q_{\text{no}}, b, 0)</td>
<td>(q_4, c, -1)</td>
<td>(q_4, x, -1)</td>
</tr>
</tbody>
</table>

\[ q_3 \]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\} \quad F = \{q_{yes}, q_{no}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{yes}, \square, 0) & (q_1, x, +1) & (q_{no}, b, 0) & (q_{no}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{no}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{no}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{no}, \square, 0) & (q_{no}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{no}, a, 0) & (q_{no}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\}\)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\Box, a, b, c, x\} \]

\[ \begin{array}{cccccc}
\delta & \Box & a & b & c & x \\
q_0 & (q_{\text{yes}}, \Box, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \Box, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \Box, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \Box, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \Box, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array} \]
Turing Machine

Language $L = \{a^n b^n c^n | n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_{\text{no}}, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[ \begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array} \]
Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\boxtimes, a, b, c, x\} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \boxtimes )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( (q_{\text{yes}}, \boxtimes, 0) )</td>
<td>( (q_1, x, +1) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_0, x, +1) )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_{\text{no}}, \boxtimes, 0) )</td>
<td>( (q_1, a, +1) )</td>
<td>( (q_2, x, +1) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_1, x, +1) )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( (q_{\text{no}}, \boxtimes, 0) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_2, b, +1) )</td>
<td>( (q_3, x, +1) )</td>
<td>( (q_2, x, +1) )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( (q_4, \boxtimes, -1) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_3, c, +1) )</td>
<td>( (q_3, x, +1) )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( (q_0, \boxtimes, +1) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_4, c, -1) )</td>
<td>( (q_4, x, -1) )</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{cccccc}
\delta & \square & a & b & c & x \\
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_{4}, \square, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \square, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>
Turing Machine

Language $L = \{a^n b^n c^n | n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ \hspace{1cm} $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ \hspace{1cm} $\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
</tr>
</tbody>
</table>

$q_4$
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>(( q_{\text{yes}}, \square, 0 ))</td>
<td>(( q_1, x, +1 ))</td>
<td>(( q_{\text{no}}, b, 0 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
<td>(( q_0, x, +1 ))</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>(( q_{\text{no}}, \square, 0 ))</td>
<td>(( q_1, a, +1 ))</td>
<td>(( q_2, x, +1 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
<td>(( q_1, x, +1 ))</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>(( q_{\text{no}}, \square, 0 ))</td>
<td>(( q_{\text{no}}, a, 0 ))</td>
<td>(( q_2, b, +1 ))</td>
<td>(( q_3, x, +1 ))</td>
<td>(( q_2, x, +1 ))</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>(( q_4, \square, -1 ))</td>
<td>(( q_{\text{no}}, a, 0 ))</td>
<td>(( q_{\text{no}}, b, 0 ))</td>
<td>(( q_3, c, +1 ))</td>
<td>(( q_3, x, +1 ))</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>(( q_0, \square, +1 ))</td>
<td>(( q_4, a, -1 ))</td>
<td>(( q_4, b, -1 ))</td>
<td>(( q_4, c, -1 ))</td>
<td>(( q_4, x, -1 ))</td>
</tr>
</tbody>
</table>
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  \quad  $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$  \quad  $\Gamma = \{\square, a, b, c, x\}$

$$
\begin{array}{c|cccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
$$

$q_4$
### Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

**\( Q \)** \( = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \)

**\( F \)** \( = \{ q_{\text{yes}}, q_{\text{no}} \} \)

**\( \Sigma \)** \( = \{ a, b, c \} \)

**\( \Gamma \)** \( = \{ \square, a, b, c, x \} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( (q_{\text{yes}}, \square, 0) )</td>
<td>( (q_1, x, +1) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_0, x, +1) )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_1, a, +1) )</td>
<td>( (q_2, x, +1) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_1, x, +1) )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_2, b, +1) )</td>
<td>( (q_3, x, +1) )</td>
<td>( (q_2, x, +1) )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( (q_4, \square, -1) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_3, c, +1) )</td>
<td>( (q_3, x, +1) )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( (q_0, \square, +1) )</td>
<td>( (q_4, a, -1) )</td>
<td>( (q_4, b, -1) )</td>
<td>( (q_4, c, -1) )</td>
<td>( (q_4, x, -1) )</td>
</tr>
</tbody>
</table>

![Turing Machine Diagram]
Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\} \quad \Gamma = \{\Box, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\Box$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \Box, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \Box, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \Box, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \Box, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \Box, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>

$q_4$
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\}$  \hspace{1em} $F = \{q_{yes}, q_{no}\}$

$\Sigma = \{a, b, c\}$  \hspace{1em} $\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{yes}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{no}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{no}, \square, 0)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \]

\[ F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \]

\[ \Gamma = \{\square, a, b, c, x\} \]

\[
\begin{array}{|c|c|c|c|c|}
\hline
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\}$  \quad $F = \{q_{yes}, q_{no}\}$

$\Sigma = \{a, b, c\}$  \quad $\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{yes}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{no}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{no}, \square, 0)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_{no}, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_{4}, \square, -1)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_{3}, c, +1)$</td>
<td>$(q_{3}, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_{0}, \square, +1)$</td>
<td>$(q_{4}, a, -1)$</td>
<td>$(q_{4}, b, -1)$</td>
<td>$(q_{4}, c, -1)$</td>
<td>$(q_{4}, x, -1)$</td>
</tr>
</tbody>
</table>
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  
$F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$  
$\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>

$q_0$
**Turing Machine**

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[
Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\}
\]

\[
\Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\}
\]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( (q_{\text{yes}}, \square, 0) )</td>
<td>( (q_1, x, +1) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_0, x, +1) )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_1, a, +1) )</td>
<td>( (q_2, x, +1) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_1, x, +1) )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_2, b, +1) )</td>
<td>( (q_3, x, +1) )</td>
<td>( (q_2, x, +1) )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( (q_4, \square, -1) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_3, c, +1) )</td>
<td>( (q_3, x, +1) )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( (q_0, \square, +1) )</td>
<td>( (q_4, a, -1) )</td>
<td>( (q_4, b, -1) )</td>
<td>( (q_4, c, -1) )</td>
<td>( (q_4, x, -1) )</td>
</tr>
</tbody>
</table>

**Diagram**

```
□ □ x x a a x x b b x x c c □ □ □
```

\[ q_0 \]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|ccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ \hspace{1cm} $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ \hspace{1cm} $\Gamma = \{\square, a, b, c, x\}$

$\delta$

| $q_0$ | $q_{\text{yes}}, \square, 0$ | $(q_1, x, +1)$ | $(q_{\text{no}}, b, 0)$ | $(q_{\text{no}}, c, 0)$ | $(q_0, x, +1)$ |
| $q_1$ | $(q_{\text{no}}, \square, 0)$ | $(q_1, a, +1)$ | $(q_2, x, +1)$ | $(q_{\text{no}}, c, 0)$ | $(q_1, x, +1)$ |
| $q_2$ | $(q_{\text{no}}, \square, 0)$ | $(q_{\text{no}}, a, 0)$ | $(q_2, b, +1)$ | $(q_3, x, +1)$ | $(q_2, x, +1)$ |
| $q_3$ | $(q_4, \square, -1)$ | $(q_{\text{no}}, a, 0)$ | $(q_{\text{no}}, b, 0)$ | $(q_3, c, +1)$ | $(q_3, x, +1)$ |
| $q_4$ | $(q_0, \square, +1)$ | $(q_4, a, -1)$ | $(q_4, b, -1)$ | $(q_4, c, -1)$ | $(q_4, x, -1)$ |

Diagram:
Turing Machine

Language $L = \{ a^n b^n c^n \mid n \geq 0 \}$

$Q = \{ q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no} \}$

$F = \{ q_{yes}, q_{no} \}$

$\Sigma = \{ a, b, c \}$

$\Gamma = \{ \square, a, b, c, x \}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{yes}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{no}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{no}, \square, 0)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n | n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
### Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th></th>
<th>( a )</th>
<th></th>
<th>( b )</th>
<th></th>
<th>( c )</th>
<th></th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_{\text{yes}}, \square, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \square, +1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, -1 )</td>
<td>( q_4, x, -1 )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of a Turing Machine with states and transitions](image-url)
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
**Turing Machine**

**Language** $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\}$  \hspace{1cm} $F = \{q_{yes}, q_{no}\}$

$\Sigma = \{a, b, c\}$  \hspace{1cm} $\Gamma = \{\square, a, b, c, x\}$

\[
\begin{array}{c|ccccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{yes}, \square, 0) & (q_1, x, +1) & (q_{no}, b, 0) & (q_{no}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{no}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{no}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{no}, \square, 0) & (q_{no}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{no}, a, 0) & (q_{no}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]

Z. Sawa (TU Ostrava)  \hspace{1cm} Theoretical Computer Science  \hspace{1cm} September 13, 2019  \hspace{1cm} 246 / 490
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ \quad $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ \quad $\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\}\]

\[\Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\}\]

\[
\begin{array}{c|ccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Language  \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
<table>
<thead>
<tr>
<th>\delta</th>
<th>\square</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(q_{\text{yes}}, \square, 0)</td>
<td>(q_1, x, +1)</td>
<td>(q_{\text{no}}, b, 0)</td>
<td>(q_{\text{no}}, c, 0)</td>
<td>(q_0, x, +1)</td>
</tr>
<tr>
<td>1</td>
<td>(q_{\text{no}}, \square, 0)</td>
<td>(q_1, a, +1)</td>
<td>(q_2, x, +1)</td>
<td>(q_{\text{no}}, c, 0)</td>
<td>(q_1, x, +1)</td>
</tr>
<tr>
<td>2</td>
<td>(q_{\text{no}}, \square, 0)</td>
<td>(q_{\text{no}}, a, 0)</td>
<td>(q_2, b, +1)</td>
<td>(q_3, x, +1)</td>
<td>(q_2, x, +1)</td>
</tr>
<tr>
<td>3</td>
<td>(q_4, \square, -1)</td>
<td>(q_{\text{no}}, a, 0)</td>
<td>(q_{\text{no}}, b, 0)</td>
<td>(q_3, c, +1)</td>
<td>(q_3, x, +1)</td>
</tr>
<tr>
<td>4</td>
<td>(q_0, \square, +1)</td>
<td>(q_4, a, -1)</td>
<td>(q_4, b, -1)</td>
<td>(q_4, c, -1)</td>
<td>(q_4, x, -1)</td>
</tr>
</tbody>
</table>
### Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_{\text{yes}, \square, -1} )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \square, +1 )</td>
<td>( q_{\text{no}, a, -1} )</td>
<td>( q_{\text{no}, b, -1} )</td>
<td>( q_{\text{no}, c, -1} )</td>
<td>( q_{4, x, -1} )</td>
</tr>
</tbody>
</table>

Diagram:

```
□ □ x x x a x x x x b x x x x c □ □ □
```

Graph:

```
q3
```

---

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019  246 / 490
### Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

**States** \( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \)

**Accepting States** \( F = \{q_{\text{yes}}, q_{\text{no}}\} \)

**Alphabet** \( \Sigma = \{a, b, c\} \)

**Input Alphabets** \( \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]

- **Diagram**: State diagram showing the transition between states for the Turing machine.
Turing Machine

Language \( L = \{a^n b^n c^n | n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]
\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_{\text{yes}}, \square, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_{\text{no}}, \square, +1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>
Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ \hspace{1cm} $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ \hspace{1cm} $\Gamma = \{\Box, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\Box$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \Box, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \Box, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \Box, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_{\text{yes}}, \Box, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \Box, +1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$  $\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_{\text{no}}, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_{\text{yes}}, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_{\text{yes}}, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_{\text{yes}}, x, -1)$</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[
\begin{align*}
Q &= \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} & \quad F &= \{q_{\text{yes}}, q_{\text{no}}\} \\
\Sigma &= \{a, b, c\} & \Gamma &= \{\square, a, b, c, x\}
\end{align*}
\]

\[
\begin{array}{c|cccccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1)
\end{array}
\]

Diagram: Turing machine with states and transitions.
Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
</tr>
</tbody>
</table>

[Diagram of Turing machine with state $q_4$]
# Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

**Q** = \{\(q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\}

**\(\Sigma\)** = \{a, b, c\} \quad **\(\Gamma\)** = \{□, a, b, c, x\}

<table>
<thead>
<tr>
<th>(\delta)</th>
<th>□</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>((q_{\text{yes}}, \square, 0))</td>
<td>((q_1, x, +1))</td>
<td>((q_{\text{no}}, b, 0))</td>
<td>((q_{\text{no}}, c, 0))</td>
<td>((q_0, x, +1))</td>
</tr>
<tr>
<td>(q_1)</td>
<td>((q_{\text{no}}, \square, 0))</td>
<td>((q_1, a, +1))</td>
<td>((q_2, x, +1))</td>
<td>((q_{\text{no}}, c, 0))</td>
<td>((q_1, x, +1))</td>
</tr>
<tr>
<td>(q_2)</td>
<td>((q_{\text{no}}, \square, 0))</td>
<td>((q_{\text{no}}, a, 0))</td>
<td>((q_2, b, +1))</td>
<td>((q_3, x, +1))</td>
<td>((q_2, x, +1))</td>
</tr>
<tr>
<td>(q_3)</td>
<td>((q_4, \square, -1))</td>
<td>((q_{\text{no}}, a, 0))</td>
<td>((q_{\text{no}}, b, 0))</td>
<td>((q_3, c, +1))</td>
<td>((q_3, x, +1))</td>
</tr>
<tr>
<td>(q_4)</td>
<td>((q_0, \square, +1))</td>
<td>((q_{\text{no}}, a, 0))</td>
<td>((q_{\text{no}}, b, 0))</td>
<td>((q_4, c, -1))</td>
<td>((q_4, x, -1))</td>
</tr>
</tbody>
</table>

---

Z. Sawa (TU Ostrava)  
Theoretical Computer Science  
September 13, 2019  
246 / 490
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$

$\Gamma = \{\square, a, b, c, x\}$

$\delta$

\[\begin{array}{c|cccc}
q & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[
Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\}
\]

\[\Sigma = \{a, b, c\} \quad \Gamma = \{\Box, a, b, c, x\}\]

\[
\begin{array}{c|cccc|cccc|c|}
\delta & \Box & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \Box, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \Box, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \Box, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_{\text{no}}, \Box, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \Box, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_{\text{no}}, x, 0 )</td>
</tr>
</tbody>
</table>

Diagram: Turing Machine in state \( q_4 \) with tape configuration \( \square \square x x x a x x x x b x x x x c \square \square \square \)
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ \quad $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ \quad $\Gamma = \{\boxminus, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\boxminus$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \boxminus, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \boxminus, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \boxminus, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_{\text{no}}, \boxminus, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_{\text{no}}, \boxminus, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_{\text{no}}, x, +1)$</td>
</tr>
</tbody>
</table>

$q_4$
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\Box, a, b, c, x\} \]

\[
\begin{array}{|c|c|c|c|c|}
\hline
\delta & \Box & a & b & c & x \\
\hline
\hline
q_0 & (q_{\text{yes}}, \Box, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \Box, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \Box, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \Box, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \Box, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4, \square, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \square, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>
Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  
$F = \{q_{\text{yes}}, q_{\text{no}}\}$

Σ = {a, b, c}  
Γ = {□, a, b, c, x}

<table>
<thead>
<tr>
<th>δ</th>
<th>□</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_0</td>
<td>(q_{\text{yes}}, □, 0)</td>
<td>(q_1, x, +1)</td>
<td>(q_{\text{no}}, b, 0)</td>
<td>(q_{\text{no}}, c, 0)</td>
<td>(q_0, x, +1)</td>
</tr>
<tr>
<td>q_1</td>
<td>(q_{\text{no}}, □, 0)</td>
<td>(q_1, a, +1)</td>
<td>(q_2, x, +1)</td>
<td>(q_{\text{no}}, c, 0)</td>
<td>(q_1, x, +1)</td>
</tr>
<tr>
<td>q_2</td>
<td>(q_{\text{no}}, □, 0)</td>
<td>(q_{\text{no}}, a, 0)</td>
<td>(q_2, b, +1)</td>
<td>(q_3, x, +1)</td>
<td>(q_2, x, +1)</td>
</tr>
<tr>
<td>q_3</td>
<td>(q_4, □, −1)</td>
<td>(q_{\text{no}}, a, 0)</td>
<td>(q_{\text{no}}, b, 0)</td>
<td>(q_3, c, +1)</td>
<td>(q_3, x, +1)</td>
</tr>
<tr>
<td>q_4</td>
<td>(q_0, □, +1)</td>
<td>(q_{\text{no}}, a, 0)</td>
<td>(q_{\text{no}}, b, 0)</td>
<td>(q_{\text{no}}, c, 0)</td>
<td>(q_4, x, −1)</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\( \Sigma = \{ a, b, c \} \quad \Gamma = \{ \square, a, b, c, x \} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}, \square, 0} )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}, b, 0} )</td>
<td>( q_{\text{no}, c, 0} )</td>
<td>( q_{\text{no}, x, +1} )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}, \square, 0} )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}, c, 0} )</td>
<td>( q_{\text{no}, x, +1} )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}, \square, 0} )</td>
<td>( q_{\text{no}, a, 0} )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_{\text{no}, x, +1} )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_{4, \square, -1} )</td>
<td>( q_{\text{no}, a, 0} )</td>
<td>( q_{\text{no}, b, 0} )</td>
<td>( q_{3, c, +1} )</td>
<td>( q_{3, x, +1} )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_{0, \square, +1} )</td>
<td>( q_{4, a, -1} )</td>
<td>( q_{4, b, -1} )</td>
<td>( q_{4, c, -1} )</td>
<td>( q_{4, x, -1} )</td>
</tr>
</tbody>
</table>
Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ 

$F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ 

$\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_{\text{yes}}, \square, 0$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>
### Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[
Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\}
\]

\[
\Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\}
\]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( (q_{\text{yes}}, \square, 0) )</td>
<td>( (q_1, x, +1) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_0, x, +1) )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_1, a, +1) )</td>
<td>( (q_2, x, +1) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_1, x, +1) )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_2, b, +1) )</td>
<td>( (q_{\text{no}}, x, +1) )</td>
<td>( (q_2, x, +1) )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( (q_4, \square, -1) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_3, c, +1) )</td>
<td>( (q_3, x, +1) )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( (q_0, \square, +1) )</td>
<td>( (q_4, a, -1) )</td>
<td>( (q_4, b, -1) )</td>
<td>( (q_4, c, -1) )</td>
<td>( (q_4, x, -1) )</td>
</tr>
</tbody>
</table>

---

| \( q_0 \) | \( \square \) | \( \square \) | \( x \) | \( x \) | \( x \) | \( x \) | \( a \) | \( x \) | \( x \) | \( x \) | \( b \) | \( x \) | \( x \) | \( x \) | \( c \) | \( \square \) | \( \square \) | \( \square \) |
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \) \quad F = \{q_{\text{yes}}, q_{\text{no}}\}

\( \Sigma = \{a, b, c\} \) \quad \Gamma = \{\Box, a, b, c, x\}

\[
\begin{array}{cccccc}
\delta & \Box & a & b & c & x \\
q_0 & (q_{\text{yes}}, \Box, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \Box, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \Box, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \Box, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \Box, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
## Turing Machine

### Language

$L = \{a^n b^n c^n \mid n \geq 0\}$

### States

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$

### Final States

$F = \{q_{\text{yes}}, q_{\text{no}}\}$

### Alphabet

$\Sigma = \{a, b, c\}$

### Tape Symbols

$\Gamma = \{\square, a, b, c, x\}$

### Transition Function

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>

### Initial State

$q_1$
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\} \quad F = \{q_{yes}, q_{no}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{yes}, \square, 0 )</td>
<td>( q_{1}, x, +1 )</td>
<td>( q_{no}, b, 0 )</td>
<td>( q_{no}, c, 0 )</td>
<td>( q_{0}, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{no}, \square, 0 )</td>
<td>( q_{1}, a, +1 )</td>
<td>( q_{2}, x, +1 )</td>
<td>( q_{no}, c, 0 )</td>
<td>( q_{1}, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{no}, \square, 0 )</td>
<td>( q_{no}, a, 0 )</td>
<td>( q_{2}, b, +1 )</td>
<td>( q_{3}, x, +1 )</td>
<td>( q_{2}, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_{4}, \square, -1 )</td>
<td>( q_{no}, a, 0 )</td>
<td>( q_{no}, b, 0 )</td>
<td>( q_{3}, c, +1 )</td>
<td>( q_{3}, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_{0}, \square, +1 )</td>
<td>( q_{4}, a, -1 )</td>
<td>( q_{4}, b, -1 )</td>
<td>( q_{4}, c, -1 )</td>
<td>( q_{4}, x, -1 )</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019  246 / 490
**Turing Machine**

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

\[
\begin{array}{|c|cccc|}
\hline
& \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]

Z. Sawa (TU Ostrava)  
Theoretical Computer Science  
September 13, 2019 246 / 490
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ \hspace{1cm} $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ \hspace{1cm} $\Gamma = \{\square, a, b, c, x\}$

\[
\begin{array}{c|ccccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\[ \Sigma = \{ a, b, c \} \quad \Gamma = \{ \square, a, b, c, x \} \]

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\Box, a, b, c, x\} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \Box )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \Box, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \Box, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \Box, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4, \Box, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \Box, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>
Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4, \square, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \square, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>

Diagram of Turing Machine:

- Initial state: \( q_3 \)
- Transition states:
  - \( q_0 \) to \( q_{\text{yes}} \)
  - \( q_1 \) to \( q_{\text{no}} \)
  - \( q_2 \) to \( q_{\text{no}} \)
  - \( q_3 \) to \( q_{4} \)
  - \( q_4 \) to \( q_0 \)
- Accepting states: \( q_{\text{yes}}, q_{\text{no}} \)
- Rejection states: \( q_{\text{no}} \)
- Blank symbol: \( \square \)
- Symbols:
  - \( x \)
  - \( a, b, c \)
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  \hspace{1cm}  $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$  \hspace{1cm}  $\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava)

Theoretical Computer Science

September 13, 2019 246 / 490
**Turing Machine**

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\Box, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \Box )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( (q_{\text{yes}}, \Box, 0) )</td>
<td>( (q_1, x, +1) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_0, x, +1) )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_{\text{no}}, \Box, 0) )</td>
<td>( (q_1, a, +1) )</td>
<td>( (q_2, x, +1) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_1, x, +1) )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( (q_{\text{no}}, \Box, 0) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_2, b, +1) )</td>
<td>( (q_3, x, +1) )</td>
<td>( (q_2, x, +1) )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( (q_4, \Box, -1) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_3, c, +1) )</td>
<td>( (q_3, x, +1) )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( (q_0, \Box, +1) )</td>
<td>( (q_{\text{no}}, x, a, -1) )</td>
<td>( (q_4, b, -1) )</td>
<td>( (q_4, c, -1) )</td>
<td>( (q_4, x, -1) )</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\sqcup, a, b, c, x\} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \sqcup )</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>(( q_{\text{yes}}, \sqcup, 0 ))</td>
<td>(( q_1, x, +1 ))</td>
<td>(( q_{\text{no}}, b, 0 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
<td>(( q_0, x, +1 ))</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>(( q_{\text{no}}, \sqcup, 0 ))</td>
<td>(( q_1, a, +1 ))</td>
<td>(( q_2, x, +1 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
<td>(( q_1, x, +1 ))</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>(( q_{\text{no}}, \sqcup, 0 ))</td>
<td>(( q_{\text{no}}, a, 0 ))</td>
<td>(( q_2, b, +1 ))</td>
<td>(( q_3, x, +1 ))</td>
<td>(( q_2, x, +1 ))</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>(( q_4, \sqcup, -1 ))</td>
<td>(( q_{\text{no}}, a, 0 ))</td>
<td>(( q_{\text{no}}, b, 0 ))</td>
<td>(( q_3, c, +1 ))</td>
<td>(( q_3, x, +1 ))</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>(( q_0, \sqcup, +1 ))</td>
<td>(( q_4, a, -1 ))</td>
<td>(( q_4, b, -1 ))</td>
<td>(( q_4, c, -1 ))</td>
<td>(( q_4, x, -1 ))</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava) Theoretical Computer Science September 13, 2019 246 / 490
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\}$$

$$\Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\}$$

$\delta$

<table>
<thead>
<tr>
<th>$q_0$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_{\text{yes}}$, $\square$, $0$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
<td></td>
</tr>
<tr>
<td>$q_{\text{no}}$, $\square$, $0$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td></td>
</tr>
<tr>
<td>$q_{\text{no}}$, $\square$, $0$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_2, x, +1)$</td>
<td></td>
</tr>
<tr>
<td>$q_4$, $\square$, $-1$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_{\text{no}}, x, +1)$</td>
<td></td>
</tr>
<tr>
<td>$q_4$, $\square$, $+1$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_{\text{no}}, x, +1)$</td>
<td></td>
</tr>
</tbody>
</table>

$q_4$
Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ \hspace{1cm} $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ \hspace{1cm} $\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>
Turing Machine

Language  \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|cccccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\} \quad F = \{q_{yes}, q_{no}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( (q_{yes}, \square, 0) )</td>
<td>( (q_1, x, +1) )</td>
<td>( (q_{no}, b, 0) )</td>
<td>( (q_{no}, c, 0) )</td>
<td>( (q_0, x, +1) )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_{no}, \square, 0) )</td>
<td>( (q_1, a, +1) )</td>
<td>( (q_2, x, +1) )</td>
<td>( (q_{no}, c, 0) )</td>
<td>( (q_1, x, +1) )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( (q_{no}, \square, 0) )</td>
<td>( (q_{no}, a, 0) )</td>
<td>( (q_2, b, +1) )</td>
<td>( (q_3, x, +1) )</td>
<td>( (q_2, x, +1) )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( (q_4, \square, -1) )</td>
<td>( (q_{no}, a, 0) )</td>
<td>( (q_{no}, b, 0) )</td>
<td>( (q_3, c, +1) )</td>
<td>( (q_3, x, +1) )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( (q_0, \square, +1) )</td>
<td>( (q_4, a, -1) )</td>
<td>( (q_4, b, -1) )</td>
<td>( (q_4, c, -1) )</td>
<td>( (q_4, x, -1) )</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\Box, a, b, c, x\} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \Box )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \Box, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \Box, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \Box, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4, \Box, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \Box, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\Box, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( q )</th>
<th>( \Box )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \Box, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \Box, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \Box, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4, \Box, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \Box, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava)

Theoretical Computer Science

September 13, 2019 246 / 490
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\}$  
$F = \{q_{yes}, q_{no}\}$

$\Sigma = \{a, b, c\}$  
$\Gamma = \{\Box, a, b, c, x\}$

\[\begin{array}{cccccc}
 q & \Delta & a & b & c & x \\
 q_0 & (q_{yes}, \Box, 0) & (q_1, x, +1) & (q_{no}, b, 0) & (q_{no}, c, 0) & (q_0, x, +1) \\
 q_1 & (q_{no}, \Box, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{no}, c, 0) & (q_1, x, +1) \\
 q_2 & (q_{no}, \Box, 0) & (q_{no}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
 q_3 & (q_4, \Box, -1) & (q_{no}, a, 0) & (q_{no}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
 q_4 & (q_0, \Box, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>(( q_{\text{yes}}, \square, 0 ))</td>
<td>(( q_1, x, +1 ))</td>
<td>(( q_{\text{no}}, b, 0 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
<td>(( q_0, x, +1 ))</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>(( q_{\text{no}}, \square, 0 ))</td>
<td>(( q_1, a, +1 ))</td>
<td>(( q_2, x, +1 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
<td>(( q_1, x, +1 ))</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>(( q_{\text{no}}, \square, 0 ))</td>
<td>(( q_{\text{no}}, a, 0 ))</td>
<td>(( q_2, b, +1 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
<td>(( q_2, x, +1 ))</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>(( q_{\text{no}}, \square, -1 ))</td>
<td>(( q_{\text{no}}, a, 0 ))</td>
<td>(( q_{\text{no}}, b, 0 ))</td>
<td>(( q_3, c, +1 ))</td>
<td>(( q_3, x, +1 ))</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>(( q_{\text{no}}, \square, +1 ))</td>
<td>(( q_{\text{no}}, a, 0 ))</td>
<td>(( q_{\text{no}}, b, 0 ))</td>
<td>(( q_4, c, -1 ))</td>
<td>(( q_4, x, -1 ))</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava)

Theoretical Computer Science

September 13, 2019 246 / 490
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad \quad F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\} \quad \quad \Gamma = \{\Box, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\Box$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \Box, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \Box, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \Box, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \Box, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \Box, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>

$Z.~Sawa~(TU~Ostrava)$

Theoretical Computer Science

September 13, 2019 246 / 490
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

\[
\begin{array}{|c|c|c|c|c|}
\hline
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]
Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\}$  \hspace{1cm} $F = \{q_{yes}, q_{no}\}$

$\Sigma = \{a, b, c\}$  \hspace{1cm} $\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{yes}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{no}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{no}, \square, 0)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>
**Turing Machine**

**Language** $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  
$F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$  
$\Gamma = \{\square, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

\[
\begin{array}{|c|cccccc|}
\hline
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \) \quad \( F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \) \quad \( \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|cccccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_{\text{no}}, \square, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\square$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, \square, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, \square, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>
Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

**Q** = \( \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \) \quad **F** = \( \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \) \quad **Γ** = \( \{\square, a, b, c, x\} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4, \square, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \square, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019  246 / 490
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

\[
\begin{array}{|c|cccc|}
\hline
\delta & \text{□} & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]
### Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[
Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \\
\Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\}
\]

<table>
<thead>
<tr>
<th>(\delta)</th>
<th>(\square) a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>((q_{\text{yes}}, \square, 0))</td>
<td>((q_1, x, +1))</td>
<td>((q_{\text{no}}, b, 0))</td>
<td>((q_{\text{no}}, c, 0))</td>
</tr>
<tr>
<td>(q_1)</td>
<td>((q_{\text{no}}, \square, 0))</td>
<td>((q_1, a, +1))</td>
<td>((q_2, x, +1))</td>
<td>((q_{\text{no}}, c, 0))</td>
</tr>
<tr>
<td>(q_2)</td>
<td>((q_{\text{no}}, \square, 0))</td>
<td>((q_{\text{no}}, a, 0))</td>
<td>((q_2, b, +1))</td>
<td>((q_3, x, +1))</td>
</tr>
<tr>
<td>(q_3)</td>
<td>((q_4, \square, -1))</td>
<td>((q_{\text{no}}, a, 0))</td>
<td>((q_{\text{no}}, b, 0))</td>
<td>((q_3, c, +1))</td>
</tr>
<tr>
<td>(q_4)</td>
<td>((q_0, \square, +1))</td>
<td>((q_4, a, -1))</td>
<td>((q_4, b, -1))</td>
<td>((q_4, c, -1))</td>
</tr>
</tbody>
</table>

---

**Diagram:**

- Initial state: \(q_0\)
- Accepting states: \(q_{\text{yes}}, q_{\text{no}}\)
- Transition rules shown in the table above.
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4, \square, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_{3, x, +1} )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \square, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>

\[ q_0 \]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q\_yes, q\_no\} \quad F = \{q\_yes, q\_no\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\box, a, b, c, x\} \)

\[
\begin{array}{ccccccc}
\delta & \box & a & b & c & x \\
q_0 & (q\_yes, \box, 0) & (q_1, x, +1) & (q\_no, b, 0) & (q\_no, c, 0) & (q_0, x, +1) \\
q_1 & (q\_no, \box, 0) & (q_1, a, +1) & (q_2, x, +1) & (q\_no, c, 0) & (q_1, x, +1) \\
q_2 & (q\_no, \box, 0) & (q\_no, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \box, -1) & (q\_no, a, 0) & (q\_no, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \box, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
### Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \square )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_{\text{yes}}, \square, 0 )</td>
<td>( q_1, x, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_0, x, +1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_1, a, +1 )</td>
<td>( q_2, x, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_1, x, +1 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_2, b, +1 )</td>
<td>( q_3, x, +1 )</td>
<td>( q_2, x, +1 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4, \square, -1 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_3, c, +1 )</td>
<td>( q_3, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_0, \square, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_4, b, -1 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>

---

![Diagram of Turing Machine](image-url)
Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  \quad  F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$  \quad  $\Gamma = \{□, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>□</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_{\text{yes}}, □, 0$</td>
<td>$(q_1, x, +1)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_{\text{no}}, □, 0$</td>
<td>$(q_1, a, +1)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_{\text{no}}, □, 0$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_2, b, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_4, □, -1)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_0, □, +1)$</td>
<td>$(q_4, a, -1)$</td>
<td>$(q_4, b, -1)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>

$q_{\text{yes}}$
Turing Machine

The diagram represents a Turing Machine with states $q_0, q_1, q_2, q_3, q_{\text{yes}}, q_{\text{no}}$. The transitions are indicated by the following rules:

- From $q_0$: $a \rightarrow x; +$ to $q_1$.
- From $q_0$: $\square; 0$ to $q_{\text{yes}}$.
- From $q_1$: $a, x; +$ to $q_4$.
- From $q_1$: $b, x; +$ to $q_2$.
- From $q_1$: $b, c; 0$ to $q_{\text{no}}$.
- From $q_2$: $b \rightarrow x; +$ to $q_4$.
- From $q_2$: $\square, c; 0$ to $q_{\text{no}}$.
- From $q_3$: $c \rightarrow x; +$ to $q_4$.
- From $q_3$: $\square, a; 0$ to $q_{\text{yes}}$.
- From $q_4$: $\square; +$ to $q_4$.
- From $q_4$: $\square; -$ to $q_4$.
- From $q_4$: $a, b, c, x; -$ to $q_4$.

The machine transitions between states based on the input symbols $a, b, c$, and the contents of the tape $x$, with the movement of the tape head indicated by $+$ and $-$.
Turing Machine – Multiplication by Three

The diagram illustrates a Turing machine that multiplies a number by three. The machine transitions between states based on the input symbols (0, 1, and □) and the current state.

- **State Diagram**:
  - **Start State**: R
  - **States**: 0, 1, 2
  - **Transitions**:
    - From R: 0, 1; +
    - From 0: 0; −, 1; −, 0, □ → 1; −
    - From 1: 1 → 0; −, 0, □ → 0; −, 1; −
    - From 2: □; −

- **Input Tape**:
  - The tape contains a sequence of symbols (□, 1, 1, 1, 0, 0, 1, 1) with a tape head at position R.
Turing Machine – Multiplication by Three

Theoretical Computer Science

September 13, 2019 248 / 490
Turing Machine – Multiplication by Three

0, 1; +

0; −

1; −

1 → 0; −

0, □ → 1; −

0, □ → 0; −

□; −
Turing Machine – Multiplication by Three

R

0, 1; +

□; −

0;

□; −

1; −

1 → 0;

0, □ → 1; −

0, □ → 0; −

1;

□ □ □

1 1 1 0 0 1 1

□ □ □

□ □ □

□ □ □

□}

Z. Sawa (TU Ostrava)

Theoretical Computer Science

September 13, 2019

248 / 490
Turing Machine – Multiplication by Three

\[
\begin{align*}
0,1;+ & \quad 0;\quad 1;\quad 1\rightarrow 0;\quad 1;\quad \\
R & \quad 0 & \quad 1 & \quad 2 \\
\quad \square;\quad \square;\quad 1 & \quad 1 & \quad 1 & \quad 0 & \quad 0 & \quad 1 & \quad 1 & \quad \square & \quad \square & \quad \square & \quad \\
\quad P & \quad \square;\quad \square;\quad \square;\quad \square;\quad 1;\quad 1;\quad 1;\quad 0;\quad 0;\quad 0;\quad 1;\quad 1;\quad \square;\quad \square;\quad \square;\quad \\
\end{align*}
\]

Z. Sawa (TU Ostrava) Theoretical Computer Science September 13, 2019 248 / 490
Turing Machine – Multiplication by Three

The diagram represents a Turing Machine configuration for multiplying a number by three. The states and transitions are as follows:

- **State R**: Move right.
- **State 0**: Move left, change 0 to 1, move right.
- **State 1**: Move right, change 1 to 0, move left.
- **State 2**: Move left, change 2 to 1, move right.

The transition rules are:

- From **State R**:
  - 0, □; + → □; −
  - □; − → □; −

- From **State 0**:
  - 0; − → □; −
  - 0, □ → 1; −

- From **State 1**:
  - 1; − → □; −
  - 0, □ → 0; −
  - 1 → 0; −

- From **State 2**:
  - 1; − → □; −

The tape contains the sequence 1 1 1 0 0 1 1.
Turing Machine – Multiplication by Three

The diagram shows a Turing machine with states R, 0, 1, and 2, and transitions based on symbols 0, 1, and □. The transitions include:

- From R on 0: 1; – → 0; –
- From 0 on 1: 1; – → 0; –
- From 0 on □: 1; – → 0; –
- From 1 on □: 1 → 0; –

The tape shows a sequence of symbols: □ □ □ 1 1 1 0 0 1 1 □ □ □ □.
Turing Machine – Multiplication by Three

0, 1; +

0; −

0, □ → 1; −

□; −

□; −

□ → 1; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −

□ → 0; −
Turing Machine – Multiplication by Three

0, 1; +

0; −

1; −

1 → 0; −

R

□; −

□; −

□; −

□; −

0, □ → 1; −

0, □ → 0; −

0, □ → 1; −

0, □ → 0; −

P

□ □ □ 1 1 1 0 0 0 1 □ □ □

2

Z. Sawa (TU Ostrava)
Theoretical Computer Science
September 13, 2019 248 / 490
Turing Machine – Multiplication by Three

0, 1; +

0; −

1; −

1 → 0; −

0, □ → 1; −

0, □ → 0; −
Turing Machine – Multiplication by Three

0, 1; +

0; −

1; −

1 → 0; −

0, □ → 1; −

0, □ → 0; −

P

R

Z. Sawa (TU Ostrava)
Theoretical Computer Science
September 13, 2019 248 / 490
Turing Machine – Multiplication by Three

0, 1; +

0; −

□; −

□; −

□; −

□; −

R

0

1

2

P

0, □ → 1; −

0, □ → 0; −

1 → 0; −

1; −

Z. Sawa (TU Ostrava)
Turing Machine – Multiplication by Three

0, 1; +

R 0 1 0

0; --

□; --

1; --

0, □ → 1; --

0, □ → 0; --

1 → 0; --

□; --

□ □ □ 1 0 1 1 0 0 1 □ □ □

2
Turing Machine – Multiplication by Three
Turing Machine – Multiplication by Three

\[ \begin{align*}
0, 1; + & \quad 0; - \\
\square; - & \quad 1; - \\
0, \square \to 1; - & \quad 1 \to 0; - \\
0, \square \to 0; - & \quad 0, \square \to 1; - \\
\end{align*} \]
Turing Machine – Multiplication by Three

0, 1; +

0; −

1; −

1 → 0; −

0, □ → 1; −

0, □ → 0; −

□; −
Turing Machine – Multiplication by Three

0, 1; +

0; −

1; −

1 → 0; −

□; −

0, □ → 1; −

0, □ → 0; −

□; −

□ 1 0 1 0 1 1 0 0 1 □ □ □ □
Nondeterministic Turing Machines

We can also consider **nondeterministic Turing machines** where for every state $q$ and symbol $b$ the transition function $\delta(q, b)$ specifies several different triples $(q', b', d)$. The machine can choose any of them.

The machine accepts a word $w$ iff it has at least one computation accepting $w$.

**Remark:** For every nondeterministic Turing machine, an equivalent deterministic Turing machine can be constructed.
A linear bounded automaton is a restricted form of a Turing machine that can use only the part of the tape where its input word is written.

There are special endmarkers denoting the left and the right end of the tape.

It is not possible to move the head to the left of the left endmarker and to the right of the right endmarker. The endmarkers cannot be overwritten.

LBA (linear bounded automata) are considered in a nondeterministic version.

The question whether the expressive power of nondeterministic LBA is the same as of deterministic LBA is an open problem.
A **generative grammar** is a tuple $G = (\Pi, \Sigma, S, P)$, where

- $\Pi$ is a finite set of nonterminals
- $\Sigma$ is a finite set of terminals, $\Pi \cap \Sigma = \emptyset$
- $S \in \Pi$ is the initial nonterminal
- $P$ is a finite set of rules of the form $\alpha \rightarrow \beta$, where $\alpha \in (\Pi \cup \Sigma)^* \Pi (\Pi \cup \Sigma)^*$ and $\beta \in (\Pi \cup \Sigma)^*$.

- $\mu_1 \alpha \mu_2 \Rightarrow \mu_1 \beta \mu_2$ if $\alpha \rightarrow \beta$ is a rule from $P$
- $L(G) = \{ w \in \Sigma^* | S \Rightarrow^* w \}$
Chomsky Hierarchy

According to forms of rules that are allowed in the grammar we can distinguish the following types of grammars:

0 – General **generative grammars** – no restrictions on the rules

1 – **Context-sensitive grammars** – rules of the form $\alpha X \beta \rightarrow \alpha \gamma \beta$, where $|\gamma| \geq 1$ (An exception is possible rule $S \rightarrow \varepsilon$, but then $S$ does not occur on the right-hand side of any rule)

2 – **Context-free grammars** – rules of the form $X \rightarrow \gamma$

3 – **Regular grammars** – rules of the form $X \rightarrow wY$ or $X \rightarrow w$

where $\alpha, \beta, \gamma \in (\Pi \cup \Sigma)^*$, $X \in \Pi$, and $w \in \Sigma^*$
Chomsky Hierarchy

According to forms of rules that are allowed in the grammar we can distinguish the following types of grammars:

0 – **General generative grammars** – no restrictions on the rules

1 – **Context-sensitive grammars** – rules of the form $\alpha X \beta \rightarrow \alpha \gamma \beta$, where $|\gamma| \geq 1$ (An exception is possible rule $S \rightarrow \varepsilon$, but then $S$ does not occur on the right-hand side of any rule)

2 – **Context-free grammars** – rules of the form $X \rightarrow \gamma$

3 – **Regular grammars** – rules of the form $X \rightarrow wY$ or $X \rightarrow w$

where $\alpha, \beta, \gamma \in (\Pi \cup \Sigma)^*$, $X \in \Pi$, and $w \in \Sigma^*$

- We say a language is of type $i$ if it is generated by a grammar of type $i$.
- Let $\mathcal{L}_i$ be the class of languages of type $i$. These classes are related as follows

$$\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$$
Chomsky Hierarchy

Names of classes of languages generated by these types of grammars:

0  – Recursively enumerable
1  – Context-sensitive
2  – Context-free
3  – Regular
Chomsky Hierarchy

Names of classes of languages generated by these types of grammars:

0  – Recursively enumerable
1  – Context-sensitive
2  – Context-free
3  – Regular

For each of these classes of languages there is a corresponding class of automata that accept precisely the languages from this class:

0  – Turing machine (deterministic or nondeterministic)
1  – Linear bounded automaton (nondeterministic, it is not known if also deterministic)
2  – Pushdown automaton (only nondeterministic)
3  – Finite automaton (deterministic or nondeterministic)
Computability and Complexity
An **algorithm** is a mechanical procedure consisting of some simple elementary steps that for any given **input** produces an **output**.

An algorithm can be described:

- in plain English
- by a pseudocode
- as a computer program in a programming language
- as a hardware circuit
- ...

Algorithms are used for solving **problems**.
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
Examples of Problems

Problem “Sorting”

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

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

**Example:**

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

**Remark:** A particular input of a problem is called an instance of the 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 $v$.

Example:
So formally, a **problem** can be defined as a tuple \((\text{In}, \text{Out}, R)\), where:

- \(\text{In}\) is the set of possible inputs
- \(\text{Out}\) is the set of possible outputs
- \(R \subseteq \text{In} \times \text{Out}\) is a relation assigning corresponding outputs to each input. This relation must satisfy

\[
\forall x \in \text{In} : \exists y \in \text{Out} : R(x, y).
\]
Encoding of Input and Output

In general, we can restrict to the case, where inputs and outputs of a problem are words over some \( \Sigma \), i.e., \( In = Out = \Sigma^* \).

Some other object (numbers, sequences of numbers, graphs, \ldots, ) then can be written (encoded) as words over this alphabet.

Example: In the problem “Sorting”, we can select as an alphabet \( \Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, , \} \).

Then an input can be for example the word

\[ 826, 13, 3901, 101, 128, 562 \]

and the output is then the word

\[ 13, 101, 128, 562, 826, 3901 \]
**Example:** If an input of some problem is for example a graph, it can be represented as a list of nodes and edges:

For example, the following graph

![Graph](image)

can be represented as word

\[(1,2,3,4,5),((1,2),(2,4),(4,3),(3,1),(1,1),(2,5),(4,5),(4,1))\]

over alphabet \(\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \_, (\_\_), (\_\_, \_)\}\).
**Remark:** Not all words from $\Sigma^*$ necessarily represent some input. We should choose such encoding that allows us to recognize easily if a word represents some input or not.
We can restrict our attention to the case where both inputs and outputs are encoded as words over alphabet \(\{0, 1\}\) (i.e., as sequences of bits).

Symbols of any other alphabet can be represented as sequences of bits.

**Example:** Alphabet \(\{a, b, c, d, e, f, g\}\)

\[
\begin{align*}
a & \leftrightarrow 001 \\
b & \leftrightarrow 010 \\
c & \leftrightarrow 011 \\
d & \leftrightarrow 100 \\
e & \leftrightarrow 101 \\
f & \leftrightarrow 110 \\
g & \leftrightarrow 111 \\
\end{align*}
\]

Word ‘defb’ can be represented as ‘100101110010’. 
Other Examples of Problems

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 situation when the set of outputs $Out$ is $\{Yes, No\}$ is quite frequent. Such problems are called **decision problems**.

We usually specify decision problems in such a way that instead describing what the output is, we introduce a question.

**Example:**

**Problem “Primality”**

- **Input:** A natural number $n$.
- **Question:** Is $n$ a prime?
Decision problem

One possibility, how the notion of a decision problem can be defined formally, is to define it as a pair $(In, T)$, where:

- $In$ is the set of all inputs,
- $T \subseteq In$ is the set of all inputs, for which the answer is $\text{YES}$. 
If we restrict to the cases where inputs are words over some alphabet $\Sigma$, then decision problems can be viewed as languages.

A language corresponding to a given decision problem is the set of those words from $\Sigma^*$ that represent inputs for which the answer is Yes.

**Example:** A language consisting of those words from $\{0, 1\}^*$ that are binary representations of primes.

For example, $101 \in L$ but $110 \not\in L$. 
An Example of a Decision Problem

SAT problem (boolean satisfiability problem)

Input: Boolean formula $\varphi$.

Question: Is $\varphi$ satisfiable?

Example:

Formula $\varphi_1 = x_1 \land (\neg x_2 \lor x_3)$ is satisfiable:
e.g., for valuation $\nu$ where $\nu(x_1) = 1$, $\nu(x_2) = 0$, $\nu(x_3) = 1$, it holds that $\nu(\varphi_1) = 1$.

Formula $\varphi_2 = (x_1 \land \neg x_1) \lor (\neg x_2 \land x_3 \land x_2)$ is not satisfiable:
for every valuation $\nu$ it holds that $\nu(\varphi_2) = 0$. 
Other special case are the so called optimization problems.

An **optimization problem** is a problem where the aim is to choose, from a set of feasible solutions, a solution that in some respect is optimal.
Optimization Problems

Other special case are the so called optimization problems.

An **optimization problem** is a problem where the aim is to choose, from a set of feasible solutions, a solution that in some respect is optimal.

**Example:** In the problem “Finding the shortest path in a graph”, the set of feasible solutions consists of all paths from the node $u$ to the node $v$. The criterion by which we compare the paths is the length of a path.
Optimization Problems

Formally, an **optimization problems** can be defined as a tuple \((\text{In}, \text{Out}, f, m, g)\), where:

- \text{In} is the set of inputs,
- \text{Out} is the set of solutions,
- \(f : \text{In} \rightarrow \mathcal{P}(\text{Out})\) is a function assigning to each input \(x\) a set of corresponding **feasible solutions** \(f(x)\),
- \(m : \bigcup_{x \in \text{In}} \{\{x\} \times f(x)\} \rightarrow \mathbb{R}\) is an **optimization function** (cost function),
- \(g\) is \text{min} or \text{max}.

The goal is to find for a given input \(x \in \text{In}\) some feasible solution \(y \in f(x)\) such that

\[ m(x, y) = g\{m(x, y') \mid y' \in f(x)\}, \]

or to find out that there is no such feasible solution for the input \(x\) (i.e., \(f(x) = \emptyset\)).
The optimization problems, where $g$ is $\text{min}$, are called \textit{minimization problems}.

The optimization problems, where $g$ is $\text{max}$, are called \textit{maximization 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?

$k = 3$
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?

$k = 3$
Examples of Problems

Independent set (IS) problem

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

**Question:** Is there an independent set of size $k$ in the graph $G$?

**Remark:** An **independent set** in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.
Examples of Problems

Independent set (IS) problem

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

**Question:** Is there an independent set of size $k$ in the graph $G$?

**Remark:** An independent set in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.
Independent Set (IS) Problem

An example of an instance where the answer is **Yes**: 

- $k = 4$

An example of an instance where the answer is **No**: 

- $k = 5$
Problem ILP (integer linear programming)

**Input:** An integer matrix $A$ and an integer vector $b$.

**Question:** Is there an integer vector $x$ such that $Ax \leq b$?

An example of an instance of the problem:

$$
A = \begin{pmatrix}
3 & -2 & 5 \\
1 & 0 & 1 \\
2 & 1 & 0 \\
\end{pmatrix} \quad b = \begin{pmatrix}
8 \\
-3 \\
5 \\
\end{pmatrix}
$$

So the question is if the following system of inequations has some integer solution:

$$
\begin{align*}
3x_1 - 2x_2 + 5x_3 & \leq 8 \\
x_1 + x_3 & \leq -3 \\
2x_1 + x_2 & \leq 5
\end{align*}
$$
One of solutions of the system

\[
\begin{align*}
3x_1 - 2x_2 + 5x_3 & \leq 8 \\
x_1 + x_3 & \leq -3 \\
2x_1 + x_2 & \leq 5
\end{align*}
\]

is for example \(x_1 = -4, \ x_2 = 1, \ x_3 = 1\), i.e.,

\[
x = \begin{pmatrix}
-4 \\
1 \\
1
\end{pmatrix}
\]

because

\[
\begin{align*}
3 \cdot (-4) - 2 \cdot 1 + 5 \cdot 1 &= -9 \leq 8 \\
-4 + 1 &= -3 \leq -3 \\
2 \cdot (-4) + 1 &= -7 \leq 5
\end{align*}
\]

So the answer for this instance is \(\text{YES}\).
Examples of Problems

Problem

**Input:** Deterministic finite automata $A_1$ and $A_2$.

**Question:** Is $L(A_1) = L(A_2)$?

Problem

**Input:** Context-free grammars $G_1$ and $G_2$.

**Question:** Is $L(G_1) = L(G_2)$?
Solutions of Problems

Solving a problem

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

1. It halts after some finite number of steps for any input of the given problem (for any input instance).
2. It produces an output from the set of possible outputs that satisfies the conditions specified in the problem statement.

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

**Remark:** correctness of an algorithm — the algorithm solves the given problem
Solutions of Problems

To each algorithm $A$ we can assign the function

$$f_A : \text{In} \rightarrow \text{Out}$$

where:

- $\text{In}$ is the set of inputs for the algorithm $A$,
- $\text{Out}$ is the set of outputs, which are generated by the algorithm $A$,
- $f_A(x)$ is the output, generated by the algorithm $A$ for an input $x \in \text{In}$.

The function need not be total (i.e., the value of $f_A(x)$ need not be defined for each $x \in \text{In}$), it can be partial:

- the value of $f_A(x)$ is undefined if the computation of the algorithm $A$ for an input $x$ never halts, if an error occurs, etc.
If we have a problem \( P = (In, Out, R) \) and an algorithm \( A \) realizing a function \( f_A : In \rightarrow Out \), we say that

\[ \text{the algorithm } A \text{ solves the problem } P \]

if:

- the value of \( f_A(x) \) is defined for each \( x \in In \),
- for each \( x \in In \) we have \( (x, f_A(x)) \in R \)
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 $P$).
A problem that is not algorithmically solvable is \textit{algorithmically unsolvable}.

A decision problem that is not decidable is \textit{undecidable}.
A **Random Access Machine (RAM)** is an idealized model of a computer.

It consists of the following parts:

- **Program unit** – contains a program for the RAM and a pointer to the currently executed instruction

- **Working memory** consists of cells numbered $0, 1, 2, \ldots$; the content of the cells can be read and written to

- **Input tape** – read-only

- **Output tape** – write-only
Random Access Machine

Program unit:

1. READ
2. JZERO 10
3. STORE *3
4. ADD 2
5. STORE 2
6. LOAD 1
7. ADD =1
8. STORE 1
9. JUMP 1
10. LOAD 2
11. DIV 1
12. STORE 2
13. LOAD =0
14. STORE 1

Input:

7 5 2 0

Working memory:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Output:

IC

ALU
Cells 0 and 1 are special and are used as *registers* of the RAM:

- **Cell 0** – a *working register* (accumulator) – the register that is one of operands for most of instructions and to which the result of most of operations is stored.

- **Cell 1** – an *index register* – it is used for indirect addressing.

Forms of *operands* of instructions \((i \in \mathbb{N})\):

<table>
<thead>
<tr>
<th>form</th>
<th>the value of the operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>(=i)</td>
<td>the number (i)</td>
</tr>
<tr>
<td>(i)</td>
<td>the number stored in the cell at address (i)</td>
</tr>
<tr>
<td>(*i)</td>
<td>the number stored in the cell at address (i + j), where (j) is the current value of the index register</td>
</tr>
</tbody>
</table>
Example:

LOAD <op>

reads the value of the operand <op> into the working register (i.e., into cell 0).

LOAD =5  – loads 5 into the working register
LOAD 5   – loads the value in cell 5 into the working register
LOAD *5  – loads the value in cell 5 + j, where j is the current value of the index register, into the working register
Random Access Machine

Input and output instructions (they have no operands):

**READ**
- the value from the cell of input tape on which is the reading head is stored into the working register and the reading head is moved right

**WRITE**
- the value in the working register is written on the output tape and the output head is moved right by one

Instructions working with memory:

**LOAD** `<op>`
- the value of the operand is loaded into the working register

**STORE** `<op>`
- the value of the operand is rewritten with the content of the working register (an operand of the form `=i` is not allowed)
Instruction for arithmetic operations:

- **ADD <op>** – the value in the working register is increased by the value of the operand (i.e., the value of the operand is added to it)
- **SUB <op>** – the value of the operand subtracted from the value of the working register
- **MUL <op>** – the value in the working register is multiplied by the value of the operand
- **DIV <op>** – the value in the working register is divided by the value of the operand (the result is truncated to an integer value)
Random Access Machine

Jump instructions:

- **JUMP <label>**  
  - the computation will continue with an instruction determined by the label

- **JZERO <label>**  
  - if the working register contains value 0, the computation will continue with an instruction determined by the label; otherwise it will continue with the following instruction

- **JGTZ <label>**  
  - if the working register contains a positive value, the computation will continue with an instruction determined by the label; otherwise it will continue with the following instruction

A halting instruction:

- **HALT**  
  - the computation is halted
Problem “Searching”

Input: An integer $x$ and a sequence of integers $a_1, a_2, \ldots, a_n$ (where $a_i \neq 0$) terminated by 0.

Output: If $a_i = x$ then the output is $i$ (if there are several such $i$ then the smallest one), otherwise the output is 0.

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found

found: LOAD 2
ADD =1
JUMP loop

output: WRITE

HALT
Random Access Machine

**start:** READ
STORE 3
LOAD =1

**loop:** STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

**found:** LOAD 2

**output:** WRITE
HALT

**Input:**
9, 13, 5, 9, 7, 2, 0

Cell 0: 0
Cell 1: 0
Cell 2: 0
Cell 3: 0
Cell 4: 0

**Output:**

**Instructions:** 0
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 0
Cell 1: 0
Cell 2: 0
Cell 3: 0
Cell 4: 0

Output:

Instructions: 0
Random Access Machine

start:  READ
    STORE  3
    LOAD   =1

loop:  STORE  2
    READ
    JZERO  output
    SUB  3
    JZERO  found
    LOAD  2
    ADD   =1
    JUMP  loop

found:  LOAD  2

output:  WRITE
    HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0:  9
Cell 1:  0
Cell 2:  0
Cell 3:  0
Cell 4:  0

Output:

Instructions: 1
start: READ
STORE 3
LOAD =1

load: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2

output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 9
Cell 1: 0
Cell 2: 0
Cell 3: 9
Cell 4: 0

Output:

Instructions: 2
Random Access Machine

start:  READ
STORE  3
LOAD   =1

loop:  STORE  2
READ
JZERO  output
SUB    3
JZERO  found
LOAD   2
ADD    =1
JUMP   loop

found:  LOAD  2
output:  WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0:  1
Cell 1:  0
Cell 2:  0
Cell 3:  9
Cell 4:  0

Output:

Instructions: 3
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input: 9, 13, 5, 9, 7, 2, 0

Cell 0: 1
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0

Output: 
Instructions: 4
Random Access Machine

```
start:   READ
STORE   3
LOAD    =1

loop:   STORE  2
READ
JZERO   output
SUB     3
JZERO   found
LOAD    2
ADD     =1
JUMP    loop

found:  LOAD   2

output: WRITE
HALT
```

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0:  13
Cell 1:  0
Cell 2:  1
Cell 3:  9
Cell 4:  0

Output:

Instructions: 5
Random Access Machine

**Input:**
9, 13, 5, 9, 7, 2, 0

**Cell:**

- Cell 0: 13
- Cell 1: 0
- Cell 2: 1
- Cell 3: 9
- Cell 4: 0

... 

**Output:**

**Instructions:** 6
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 4
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0

Output:

Instructions: 7
Random Access Machine

start:   READ
STORE   3
LOAD    =1

loop:   STORE  2
READ
JZERO   output
SUB     3
JZERO   found
LOAD    2
ADD     =1
JUMP    loop

found:  LOAD  2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 4
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0

Output:

Instructions: 8
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 1
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0

Output:

Instructions: 9
Random Access Machine

start:   READ
         STORE 3
         LOAD =1

loop:   STORE 2
         READ
         JZERO output
         SUB 3
         JZERO found
         LOAD 2
         ADD =1
         JUMP loop

found:  LOAD 2

output: WRITE
         HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0:  2
Cell 1:  0
Cell 2:  1
Cell 3:  9
Cell 4:  0

Output:

Instructions: 10
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 2
Cell 1: 0
Cell 2: 1
Cell 3: 9
Cell 4: 0

Output:

Instructions: 11
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 2
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0

Output:

Instructions: 12
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input: 9, 13, 5, 9, 7, 2, 0
Cell 0: 5
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0

Output: Instructions: 13
Random Access Machine

\[
\begin{align*}
\text{start:} & \quad \text{READ} \\
& \quad \text{STORE} \quad 3 \\
& \quad \text{LOAD} \quad =1 \\
\text{loop:} & \quad \text{STORE} \quad 2 \\
& \quad \text{READ} \\
& \quad \text{JZERO} \quad \text{output} \\
& \quad \text{SUB} \quad 3 \\
& \quad \text{JZERO} \quad \text{found} \\
& \quad \text{LOAD} \quad 2 \\
& \quad \text{ADD} \quad =1 \\
& \quad \text{JUMP} \quad \text{loop} \\
\text{found:} & \quad \text{LOAD} \quad 2 \\
\text{output:} & \quad \text{WRITE} \\
& \quad \text{HALT}
\end{align*}
\]

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 5
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0

Output:
Instructions: 14
Random Access Machine

start:  READ
       STORE  3
       LOAD   =1
loop:  STORE  2
       READ
       JZERO output
       SUB    3
       JZERO found
       LOAD   2
       ADD    =1
       JUMP   loop
found: LOAD   2
output: WRITE
       HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: -4
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0

Output:

Instructions: 15
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2

output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: -4
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0

Output:

Instructions: 16
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 2
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0

Output:

Instructions: 17
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 3
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0

Output:

Instructions: 18
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 3
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0

Output:

Instructions: 19
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 3
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0

Output:

Instructions: 20
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 9
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0

Output:

Instructions: 21
Random Access Machine

start: READ
  STORE 3
  LOAD =1

loop: STORE 2
  READ
  JZERO output
  SUB 3
  JZERO found
  LOAD 2
  ADD =1
  JUMP loop

found: LOAD 2

output: WRITE

HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 9
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0

Output:

Instructions: 22
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2

output: WRITE
HALT

Input:

9, 13, 5, 9, 7, 2, 0

Cell 0: 0
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0

Output:

Instructions: 23
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 0
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0
...

Output:

Instructions: 24
start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 3
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0

Output:

Instructions: 25
Random Access Machine

start: READ
STORE 3
LOAD =1

loop: STORE 2
READ
JZERO output
SUB 3
JZERO found
LOAD 2
ADD =1
JUMP loop

found: LOAD 2
output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 3
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0

Output: 3
Instructions: 26
Random Access Machine

start:  READ
STORE  3
LOAD   =1

loop:  STORE  2
READ
JZERO  output
SUB    3
JZERO  found
LOAD   2
ADD    =1
JUMP   loop

found: LOAD  2

output: WRITE
HALT

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0:  3
Cell 1:  0
Cell 2:  3
Cell 3:  9
Cell 4:  0

Output:  3

Instructions: 27
Church-Turing thesis

Every algorithm can be implemented as a Turing machine.

It is not a theorem that can be proved in a mathematical sense – it is not formally defined what an algorithm is.

The thesis was formulated in 1930s independently by Alan Turing and Alonzo Church.
Examples of mathematical formalisms modelling the notion of an algorithm:

- Random Access Machines
- Turing machines
- Lambda calculus
- Recursive functions
- ...

We can also mention:

- An arbitrary (general purpose) programming language (for example C, Java, Lisp, Haskell, Prolog, etc.).

All these models are equivalent with respect to algorithms that can be implemented by them.
Algorithms
Example: An algorithm described by pseudocode:

Algorithm 1: An algorithm for finding the maximal element in an array

1 \textbf{Find-Max} \((A, n)\):  
2 \textbf{begin}  
3 \hspace{1em} k := 0  
4 \hspace{1em} \textbf{for } i := 1 \textbf{ to } n - 1 \textbf{ do}  
5 \hspace{2em} \textbf{if } A[i] > A[k] \textbf{ then}  
6 \hspace{3em} k := i  
7 \hspace{1em} \textbf{end}  
8 \textbf{end}  
9 \textbf{return } A[k]  
10 \textbf{end}
Algorithm

- processes an **input**
- generates an **output**

From the point of view of an analysis how a given algorithm works, it usually makes only a little difference if the algorithm:
- reads input data from some input device (e.g., from a file, from a keyboard, etc.)
- writes data to some output device (e.g., to a file, on a screen, etc.)

or
- reads input data from a memory (e.g., they are given to it as parameters)
- writes data somewhere to memory (e.g., it returns them as a return value)
Instructions can be roughly divided into two groups:

- instructions working directly with data:
  - assignment
  - evaluation of values of expressions in conditions
  - reading input, writing output
  - ...

- instructions affecting the control flow — they determine, which instructions will be executed, in what order, etc.:
  - branching (if, switch, ...)
  - cycles (while, do .. while, for, ...)
  - organisation of instructions into blocks
  - returns from subprograms (return, ...)
  - ...

Z. Sawa (TU Ostrava) Theoretical Computer Science September 13, 2019 297 / 490
Control Flow Graph

0

1

2

3

4

5

6

7

\[ k := 0 \]

\[ i := 1 \]

\[ i := i + 1 \]

\[ \text{result} := A[k] \]

\[ [i \geq n] \]

\[ [i < n] \]

\[ [A[i] > A[k]] \]

\[ [A[i] \leq A[k]] \]

\[ k := i \]
Some Basic Constructions of Structured Programming

\[ S_1 \]
\[ S_2 \]

\[ S_1; S_2 \]

If \( B \) then \( S_1 \) else \( S_2 \)

If \( B \) then \( S \)
Some Basic Constructions of Structured Programming

while $B$ do $S$

$S$

do $S$ while $B$

$S$

[\neg B] [B]
Some Basic Constructions of Structured Programming

for $i := a$ to $b$ do $S$

while $i \leq b$ do
    $S$
    $i := i + 1$
end
Short-circuit evaluation of compound conditions, e.g.:

\[
\text{while } i < n \text{ and } A[i] > x \text{ do } \ldots
\]
Control-flow Realized by GOTO

- **goto ℓ** — unconditional jump
- **if B then goto ℓ** — conditional jump

Example:

0: \( k := 0 \)
1: \( i := 1 \)
2: \( \text{goto 6} \)
3: \( \text{if } A[i] \leq A[k] \text{ then goto 5} \)
4: \( k := i \)
5: \( i := i + 1 \)
6: \( \text{if } i < n \text{ then goto 3} \)
7: \( \text{return } A[k] \)
Control-flow Realized by GOTO

- **goto** \( \ell \) — unconditional jump
- **if** \( B \) then **goto** \( \ell \) — conditional jump

Example:

```
start:  \( k := 0 \)
        \( i := 1 \)
        **goto** L3
L1:    **if** \( A[i] \leq A[k] \) then **goto** L2
        \( k := i \)
L2:    \( i := i + 1 \)
L3:    **if** \( i < n \) then **goto** L1
        return \( A[k] \)
```
Evaluation of a complicated expression such as

\[ A[i + s] := (B[3 \times j + 1] + x) \times y + 8 \]

can be replaced by a sequence of simpler instructions on the lower level, such as

\[
\begin{align*}
t_1 & := i + s \\
t_2 & := 3 \times j \\
t_2 & := t_2 + 1 \\
t_3 & := B[t_2] \\
t_3 & := t_3 + x \\
t_3 & := t_3 \times y \\
t_3 & := t_3 + 8 \\
A[t_1] & := t_3
\end{align*}
\]
Computation of an Algorithm

An algorithm is executed by a machine — it can be for example:
- real computer — executes instructions of a machine code
- virtual machine — executes instructions of a bytecode
- some idealized mathematical model of a computer
- ...

The machine can be:
- specialized — executes only one algorithm
- universal — can execute arbitrary algorithm, given in a form of program

The machine performs steps.

The algorithm processes a particular input during its computation.
During a computation, the machine must remember:

- the current instruction
- the content of its working memory

It depends on the type of the machine:

- what is the type of data, with which the machine works
- how this data are organized in its memory

Depending on the type of the algorithm and the type of analysis, which we want to do, we can decide if it makes sense to include in memory also the places

- from which the input data are read
- where the output data are written
**Configuration** — the description of the global state of the machine in some particular step during a computation

**Example:** A configuration of the form

\[(q, mem)\]

where

- \(q\) — the current control state
- \(mem\) — the current content of memory of the machine — the values assigned currently to variables.

An example of a content of memory \(mem\):

\[\langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle\]
Computation of an Algorithm

An example of a configuration:

\[(2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ?\rangle)\]

A **computation** of a machine $M$ executing an algorithm $Alg$, where it processes an input $w$, in a sequence of configurations.

- It starts in an **initial configuration**.
- In every step, it goes from one configuration to another.
- The computation ends in a **final configuration**.
Computation of an Algorithm

\[ k := 0 \]
\[ i := 1 \]
\[ i < n \]
\[ i \geq n \]
\[ \begin{align*}
   & A[i] \leq A[k]
\end{align*} \]
\[ k := i \]
\[ i := i + 1 \]
\[ \text{result} := A[k] \]
Example: A computation, where algorithm \texttt{FIND-MAX} processes an input where \( A = [3, 8, 1, 3, 6] \) and \( n = 5 \).

\[
\alpha_0: \langle 0, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: ?, \ result: \ ? \rangle \rangle
\]
**Example:** A computation, where algorithm `FIND-MAX` processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

\[
\alpha_0: (0, \langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: \rangle)\\
\alpha_1: (1, \langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: \rangle)
\]
Example: A computation, where algorithm **Find-Max** processes an input where \( A = [3, 8, 1, 3, 6] \) and \( n = 5 \).

\[
\begin{align*}
\alpha_0 & : (0, \langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle) \\
\alpha_1 & : (1, \langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle) \\
\alpha_2 & : (2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle)
\end{align*}
\]
Example: A computation, where algorithm **FIND-MAX** processes an input where \( A = [3, 8, 1, 3, 6] \) and \( n = 5 \).

\[
\alpha_0: (0, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: ?, \ result: ?\rangle) \\
\alpha_1: (1, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: 0, \ result: ?\rangle) \\
\alpha_2: (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ?\rangle) \\
\alpha_3: (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ?\rangle)
\]
Example: A computation, where algorithm $\text{FIND-MAX}$ processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

$$
\alpha_0: (0, \langle A: [3, 8, 1, 3, 6], \hspace{0.2cm} n: 5, \hspace{0.2cm} i: ?, \hspace{0.2cm} k: ?, \hspace{0.2cm} \text{result: ?} \rangle)
$$

$$
\alpha_1: (1, \langle A: [3, 8, 1, 3, 6], \hspace{0.2cm} n: 5, \hspace{0.2cm} i: ?, \hspace{0.2cm} k: 0, \hspace{0.2cm} \text{result: ?} \rangle)
$$

$$
\alpha_2: (2, \langle A: [3, 8, 1, 3, 6], \hspace{0.2cm} n: 5, \hspace{0.2cm} i: 1, \hspace{0.2cm} k: 0, \hspace{0.2cm} \text{result: ?} \rangle)
$$

$$
\alpha_3: (3, \langle A: [3, 8, 1, 3, 6], \hspace{0.2cm} n: 5, \hspace{0.2cm} i: 1, \hspace{0.2cm} k: 0, \hspace{0.2cm} \text{result: ?} \rangle)
$$

$$
\alpha_4: (4, \langle A: [3, 8, 1, 3, 6], \hspace{0.2cm} n: 5, \hspace{0.2cm} i: 1, \hspace{0.2cm} k: 0, \hspace{0.2cm} \text{result: ?} \rangle)
$$
Example: A computation, where algorithm \texttt{FIND-MAX} processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

\begin{align*}
\alpha_0: \langle 0, \langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle \rangle \\
\alpha_1: \langle 1, \langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle \rangle \\
\alpha_2: \langle 2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle \rangle \\
\alpha_3: \langle 3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle \rangle \\
\alpha_4: \langle 4, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle \rangle \\
\alpha_5: \langle 5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle \rangle \\
\end{align*}
**Example:** A computation, where algorithm $\text{FIND-MAX}$ processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

$$
\begin{align*}
\alpha_0: & \ (0, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: \ ?, \ k: \ ?, \ result: \ ? \rangle) \\
\alpha_1: & \ (1, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: \ ?, \ k: 0, \ result: \ ? \rangle) \\
\alpha_2: & \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: \ ? \rangle) \\
\alpha_3: & \ (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: \ ? \rangle) \\
\alpha_4: & \ (4, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: \ ? \rangle) \\
\alpha_5: & \ (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 1, \ result: \ ? \rangle) \\
\alpha_6: & \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: \ ? \rangle)
\end{align*}
$$
Example: A computation, where algorithm \textbf{FIND-MAX} processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

\begin{align*}
\alpha_0: \ (0, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: ?, \ result: ? \rangle) \\
\alpha_1: \ (1, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: 0, \ result: ? \rangle) \\
\alpha_2: \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_3: \ (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_4: \ (4, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_5: \ (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 1, \ result: ? \rangle) \\
\alpha_6: \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_7: \ (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle)
\end{align*}
**Example:** A computation, where algorithm \texttt{FIND-MAX} processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

\[
\begin{align*}
\alpha_0 & : (0, \langle A: [3, 8, 1, 3, 6], n: 5, i: \?, k: \?, \text{result}: \? \rangle) \\
\alpha_1 & : (1, \langle A: [3, 8, 1, 3, 6], n: 5, i: \?, k: 0, \text{result}: \? \rangle) \\
\alpha_2 & : (2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, \text{result}: \? \rangle) \\
\alpha_3 & : (3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, \text{result}: \? \rangle) \\
\alpha_4 & : (4, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, \text{result}: \? \rangle) \\
\alpha_5 & : (5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, \text{result}: \? \rangle) \\
\alpha_6 & : (2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, \text{result}: \? \rangle) \\
\alpha_7 & : (3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, \text{result}: \? \rangle) \\
\alpha_8 & : (5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, \text{result}: \? \rangle)
\end{align*}
\]
Example: A computation, where algorithm **FIND-MAX** processes an input where \( A = [3, 8, 1, 3, 6] \) and \( n = 5 \).

\[
\begin{align*}
\alpha_0: & \ (0, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: ?, \ result: ? \rangle) \\
\alpha_1: & \ (1, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: 0, \ result: ? \rangle) \\
\alpha_2: & \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_3: & \ (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_4: & \ (4, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_5: & \ (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 1, \ result: ? \rangle) \\
\alpha_6: & \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 1, \ result: ? \rangle) \\
\alpha_7: & \ (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_8: & \ (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_9: & \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 3, \ k: 1, \ result: ? \rangle)
\end{align*}
\]
Example: A computation, where algorithm \texttt{FIND-MAX} processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

\begin{align*}
\alpha_0: & \ (0, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: ?, \ result: ? \rangle) \\
\alpha_1: & \ (1, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: 0, \ result: ? \rangle) \\
\alpha_2: & \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_3: & \ (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_4: & \ (4, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_5: & \ (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 1, \ result: ? \rangle) \\
\alpha_6: & \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 1, \ result: ? \rangle) \\
\alpha_7: & \ (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_8: & \ (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_9: & \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 3, \ k: 1, \ result: ? \rangle) \\
\alpha_{10}: & \ (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 3, \ k: 1, \ result: ? \rangle)
\end{align*}
Example: A computation, where algorithm `FIND-MAX` processes an input where \( A = [3, 8, 1, 3, 6] \) and \( n = 5 \).

\[
\begin{align*}
\alpha_0: & \quad (0, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: ?, \ result: ? \rangle) \\
\alpha_1: & \quad (1, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: 0, \ result: ? \rangle) \\
\alpha_2: & \quad (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_3: & \quad (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_4: & \quad (4, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_5: & \quad (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 1, \ result: ? \rangle) \\
\alpha_6: & \quad (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 1, \ result: ? \rangle) \\
\alpha_7: & \quad (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_8: & \quad (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_9: & \quad (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_{10}: & \quad (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 3, \ k: 1, \ result: ? \rangle) \\
\alpha_{11}: & \quad (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 3, \ k: 1, \ result: ? \rangle)
\end{align*}
\]
Example: A computation, where algorithm \texttt{FIND-MAX} processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

\begin{align*}
\alpha_0: & \quad (0, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: ?, \quad k: ?, \quad \text{result: ?} \rangle) \\
\alpha_1: & \quad (1, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: ?, \quad k: 0, \quad \text{result: ?} \rangle) \\
\alpha_2: & \quad (2, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: 1, \quad k: 0, \quad \text{result: ?} \rangle) \\
\alpha_3: & \quad (3, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: 1, \quad k: 0, \quad \text{result: ?} \rangle) \\
\alpha_4: & \quad (4, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: 1, \quad k: 0, \quad \text{result: ?} \rangle) \\
\alpha_5: & \quad (5, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: 1, \quad k: 1, \quad \text{result: ?} \rangle) \\
\alpha_6: & \quad (2, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: 1, \quad k: 1, \quad \text{result: ?} \rangle) \\
\alpha_7: & \quad (3, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: 2, \quad k: 1, \quad \text{result: ?} \rangle) \\
\alpha_8: & \quad (5, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: 2, \quad k: 1, \quad \text{result: ?} \rangle) \\
\alpha_9: & \quad (2, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: 2, \quad k: 1, \quad \text{result: ?} \rangle) \\
\alpha_{10}: & \quad (3, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: 3, \quad k: 1, \quad \text{result: ?} \rangle) \\
\alpha_{11}: & \quad (5, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: 3, \quad k: 1, \quad \text{result: ?} \rangle) \\
\alpha_{12}: & \quad (2, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: 4, \quad k: 1, \quad \text{result: ?} \rangle)
\end{align*}
Computation of an Algorithm

Example: A computation, where algorithm FIND-MAX processes an input where \( A = [3, 8, 1, 3, 6] \) and \( n = 5 \).

\[
\begin{align*}
\alpha_0 & : (0, \langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle) \\
\alpha_1 & : (1, \langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle) \\
\alpha_2 & : (2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle) \\
\alpha_3 & : (3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle) \\
\alpha_4 & : (4, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle) \\
\alpha_5 & : (5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle) \\
\alpha_6 & : (2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle) \\
\alpha_7 & : (3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle) \\
\alpha_8 & : (5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle) \\
\alpha_9 & : (2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle) \\
\alpha_{10} & : (3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle) \\
\alpha_{11} & : (5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle) \\
\alpha_{12} & : (2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle) \\
\alpha_{13} & : (3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle)
\end{align*}
\]
Example: A computation, where algorithm \texttt{FIND-MAX} processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

\begin{align*}
\alpha_0: & \ (0, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: ?, \ result: ? \rangle) \\
\alpha_1: & \ (1, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: 0, \ result: ? \rangle) \\
\alpha_2: & \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_3: & \ (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_4: & \ (4, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_5: & \ (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 1, \ result: ? \rangle) \\
\alpha_6: & \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_7: & \ (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_8: & \ (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_9: & \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 3, \ k: 1, \ result: ? \rangle) \\
\alpha_{10}: & \ (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 3, \ k: 1, \ result: ? \rangle) \\
\alpha_{11}: & \ (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 3, \ k: 1, \ result: ? \rangle) \\
\alpha_{12}: & \ (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 4, \ k: 1, \ result: ? \rangle) \\
\alpha_{13}: & \ (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 4, \ k: 1, \ result: ? \rangle) \\
\alpha_{14}: & \ (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 4, \ k: 1, \ result: ? \rangle)
\end{align*}
**Example:** A computation, where algorithm **Find-Max** processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

$\alpha_0$: $(0, \langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, \text{result: ?} \rangle)$
$\alpha_1$: $(1, \langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, \text{result: ?} \rangle)$
$\alpha_2$: $(2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, \text{result: ?} \rangle)$
$\alpha_3$: $(3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, \text{result: ?} \rangle)$
$\alpha_4$: $(4, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, \text{result: ?} \rangle)$
$\alpha_5$: $(5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, \text{result: ?} \rangle)$
$\alpha_6$: $(2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, \text{result: ?} \rangle)$
$\alpha_7$: $(3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, \text{result: ?} \rangle)$
$\alpha_8$: $(5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, \text{result: ?} \rangle)$
$\alpha_9$: $(2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, \text{result: ?} \rangle)$
$\alpha_{10}$: $(3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, \text{result: ?} \rangle)$
$\alpha_{11}$: $(5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, \text{result: ?} \rangle)$
$\alpha_{12}$: $(2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, \text{result: ?} \rangle)$
$\alpha_{13}$: $(3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, \text{result: ?} \rangle)$
$\alpha_{14}$: $(5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, \text{result: ?} \rangle)$
$\alpha_{15}$: $(2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 5, k: 1, \text{result: ?} \rangle)$
**Example:** A computation, where algorithm **FIND-MAX** processes an input where \( A = [3, 8, 1, 3, 6] \) and \( n = 5 \).

\[
\begin{align*}
\alpha_0: & \quad \langle 0, \langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, \text{result: ?} \rangle \rangle \\
\alpha_1: & \quad \langle 1, \langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, \text{result: ?} \rangle \rangle \\
\alpha_2: & \quad \langle 2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, \text{result: ?} \rangle \rangle \\
\alpha_3: & \quad \langle 3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, \text{result: ?} \rangle \rangle \\
\alpha_4: & \quad \langle 4, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, \text{result: ?} \rangle \rangle \\
\alpha_5: & \quad \langle 5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, \text{result: ?} \rangle \rangle \\
\alpha_6: & \quad \langle 2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, \text{result: ?} \rangle \rangle \\
\alpha_7: & \quad \langle 3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, \text{result: ?} \rangle \rangle \\
\alpha_8: & \quad \langle 5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, \text{result: ?} \rangle \rangle \\
\alpha_9: & \quad \langle 2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, \text{result: ?} \rangle \rangle \\
\alpha_{10}: & \quad \langle 3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, \text{result: ?} \rangle \rangle \\
\alpha_{11}: & \quad \langle 5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, \text{result: ?} \rangle \rangle \\
\alpha_{12}: & \quad \langle 2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, \text{result: ?} \rangle \rangle \\
\alpha_{13}: & \quad \langle 3, \langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, \text{result: ?} \rangle \rangle \\
\alpha_{14}: & \quad \langle 5, \langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, \text{result: ?} \rangle \rangle \\
\alpha_{15}: & \quad \langle 2, \langle A: [3, 8, 1, 3, 6], n: 5, i: 5, k: 1, \text{result: ?} \rangle \rangle \\
\alpha_{16}: & \quad \langle 6, \langle A: [3, 8, 1, 3, 6], n: 5, i: 5, k: 1, \text{result: ?} \rangle \rangle
\end{align*}
\]
**Example:** A computation, where algorithm **Find-Max** processes an input where $A = [3, 8, 1, 3, 6]$ and $n = 5$.

$$
\begin{align*}
\alpha_0 &: (0, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: ?, \ result: ? \rangle) \\
\alpha_1 &: (1, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: ?, \ k: 0, \ result: ? \rangle) \\
\alpha_2 &: (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_3 &: (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_4 &: (4, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 0, \ result: ? \rangle) \\
\alpha_5 &: (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 1, \ k: 1, \ result: ? \rangle) \\
\alpha_6 &: (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_7 &: (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_8 &: (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 2, \ k: 1, \ result: ? \rangle) \\
\alpha_9 &: (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 3, \ k: 1, \ result: ? \rangle) \\
\alpha_{10} &: (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 3, \ k: 1, \ result: ? \rangle) \\
\alpha_{11} &: (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 3, \ k: 1, \ result: ? \rangle) \\
\alpha_{12} &: (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 4, \ k: 1, \ result: ? \rangle) \\
\alpha_{13} &: (3, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 4, \ k: 1, \ result: ? \rangle) \\
\alpha_{14} &: (5, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 4, \ k: 1, \ result: ? \rangle) \\
\alpha_{15} &: (2, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 5, \ k: 1, \ result: ? \rangle) \\
\alpha_{16} &: (6, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 5, \ k: 1, \ result: ? \rangle) \\
\alpha_{17} &: (7, \langle A: [3, 8, 1, 3, 6], \ n: 5, \ i: 5, \ k: 1, \ result: 8 \rangle)
\end{align*}
$$
Computation of an Algorithm

By executing an instruction $I$, the machine goes from configuration $\alpha$ to configuration $\alpha'$:

$$\alpha \xrightarrow{I} \alpha'$$

A computation can be:

- **Finite:**

$$\alpha_0 \xrightarrow{l_0} \alpha_1 \xrightarrow{l_1} \alpha_2 \xrightarrow{l_2} \alpha_3 \xrightarrow{l_3} \alpha_4 \xrightarrow{l_4} \cdots \xrightarrow{l_{t-2}} \alpha_{t-1} \xrightarrow{l_{t-1}} \alpha_t$$

where $\alpha_t$ is a final configuration.

- **Infinite:**

$$\alpha_0 \xrightarrow{l_0} \alpha_1 \xrightarrow{l_1} \alpha_2 \xrightarrow{l_2} \alpha_3 \xrightarrow{l_3} \alpha_4 \xrightarrow{l_4} \cdots$$
A computation can be described in two different ways:

- as a sequence of configurations $\alpha_0, \alpha_1, \alpha_2, \ldots$
- as a sequence of executed instructions $l_0, l_1, l_2, \ldots$
Algorithms are used for solving problems.

- **Problem** — a specification **what** should be computed by an algorithm:
  - Description of inputs
  - Description of outputs
  - How outputs are related to inputs

- **Algorithm** — a particular procedure that describes **how** to compute an output for each possible input
Example: The problem of finding a maximal element in an array:

**Input:** An array $A$ indexed from zero and a number $n$ representing the number of elements in array $A$. It is assumed that $n \geq 1$.

**Output:** A value $result$ of a maximal element in the array $A$, i.e., the value $result$ such that:

- $A[j] \leq result$ for all $j \in \mathbb{N}$, where $0 \leq j < n$, and
- there exists $j \in \mathbb{N}$ such that $0 \leq j < n$ and $A[j] = result$.

An instance of a problem — concreate input data, e.g.,

$$A = [3, 8, 1, 3, 6], \quad n = 5.$$  

The output for this instance is value $8$. 
Correctness of Algorithms

Definition

An algorithm $\text{Alg}$ solves a given problem $P$, if for each instance $w$ of problem $P$, the following conditions are satisfied:

- The computation of algorithm $\text{Alg}$ on input $w$ halts after finite number of steps.
- Algorithm $\text{Alg}$ generates a correct output for input $w$ according to conditions in problem $P$.

An algorithm that solves problem $P$ is a correct solution of this problem.
Algorithm $\text{Alg}$ is not a correct solution of problem $P$ if there exists an input $w$ such that in the computation on this input, one of the following incorrect behaviours occurs:

- some incorrect illegal operation is performed (an access to an element of an array with index out of bounds, division by zero, \ldots),
- the generated output does not satisfy the conditions specified in problem $P$,
- the computation never halts.

**Testing** — running the algorithm with different inputs and checking whether the algorithm behaves correctly on these inputs.

Testing can be used to show the presence of bugs but not to show that algorithm behaves correctly for all inputs.
Correctness of Algorithms

Generally, it is reasonable to divide a proof of correctness of an algorithm into two parts:

- Showing that the algorithm never does anything “wrong” for any input:
  - no illegal operation is performed during a computation
  - if the program halts, the generated output will be “correct”

- Showing that for every input the algorithm halts after a finite number of steps.
Invariant — a condition that must be always satisfied in a given position in a code of the algorithm (i.e., in all possible computations for all allowed inputs) whenever the algorithm goes through this position.

We say that a configuration $\alpha$ is reachable if there exists an input $w$ such that $\alpha$ is one of configurations through which the algorithm goes in the computation on input $w$.

If an algorithm is represented by a control-flow graph, for a given control state $q$ (i.e., a node of the graph) we can specify invariants that hold in every reachable configuration with control state $q$. 
Invariants can be written as formulas of predicate logic:

- **free** variables correspond to variables of the program

- a **valuation** is determined by values of program variables in a given configuration

**Example:** Formula

\[(1 \leq i) \land (i \leq n)\]

holds for example in a configuration where variable \(i\) has value 5 and variable \(n\) has value 14.
Invariants

\[ k := 0 \]
\[ i := 1 \]
\[ i < n \]
\[ A[i] > A[k] \]
\[ i := i + 1 \]
\[ A[i] \leq A[k] \]
\[ k := i \]
\[ i \geq n \]

result := A[k]

Z. Sawa (TU Ostrava)
Theoretical Computer Science
September 13, 2019 320 / 490
Examples of invariants:
- an invariant in a control state \( q \) is represented by a formula \( \varphi_q \)

Invariants for individual control states (so far only hypotheses):
- \( \varphi_0: (n \geq 1) \)
- \( \varphi_1: (n \geq 1) \land (k = 0) \)
- \( \varphi_2: (n \geq 1) \land (1 \leq i \leq n) \land (0 \leq k < i) \)
- \( \varphi_3: (n \geq 1) \land (1 \leq i < n) \land (0 \leq k < i) \)
- \( \varphi_4: (n \geq 1) \land (1 \leq i < n) \land (0 \leq k < i) \)
- \( \varphi_5: (n \geq 1) \land (1 \leq i < n) \land (0 \leq k \leq i) \)
- \( \varphi_6: (n \geq 1) \land (i = n) \land (0 \leq k < n) \)
- \( \varphi_7: (n \geq 1) \land (i = n) \land (0 \leq k < n) \)
Invariants

Examples of invariants:

- an invariant in a control state \( q \) is represented by a formula \( \varphi_q \)

Invariants for individual control states (so far only hypotheses):

- \( \varphi_0: \ n \geq 1 \)
- \( \varphi_1: \ n \geq 1, \ k = 0 \)
- \( \varphi_2: \ n \geq 1, \ 1 \leq i \leq n, \ 0 \leq k < i \)
- \( \varphi_3: \ n \geq 1, \ 1 \leq i < n, \ 0 \leq k < i \)
- \( \varphi_4: \ n \geq 1, \ 1 \leq i < n, \ 0 \leq k < i \)
- \( \varphi_5: \ n \geq 1, \ 1 \leq i < n, \ 0 \leq k \leq i \)
- \( \varphi_6: \ n \geq 1, \ i = n, \ 0 \leq k < n \)
- \( \varphi_7: \ n \geq 1, \ i = n, \ 0 \leq k < n \)
Invariants

Checking that the given invariants really hold:

- It is necessary to check for each instruction of the algorithm that under the assumption that a specified invariant holds before an execution of the instruction, the other specified invariant holds after the execution of the instruction.

Let us assume the algorithm is represented as a control-flow graph:

- edges correspond to instructions
- consider an edge from state $q$ to state $q'$ labelled with instruction $I$
- let us say that (so far non-verified) invariants for states $q$ and $q'$ are expressed by formulas $\varphi$ and $\varphi'$
- for this edge we must check that for every configurations $\alpha = (q, \text{mem})$ and $\alpha' = (q', \text{mem}')$ such that $\alpha \xrightarrow{I} \alpha'$, it holds that if $\varphi$ holds in configuration $\alpha$, then $\varphi'$ holds in configuration $\alpha'$
Invariants

Checking instructions, which are conditional tests:

- an edge labelled with a conditional test \([B]\)

A content of memory is not modified. It is sufficient to check that the following implication holds

\[(\varphi \land B) \rightarrow \varphi'\]

**Remark:** The given implication must hold for all possible values of variables.

**Example:** Let us assume that formulas contain only variables \(n, i, k\), and that values of these variables are integers:

\[(\forall n \in \mathbb{Z})(\forall i \in \mathbb{Z})(\forall k \in \mathbb{Z}) (\varphi \land B \rightarrow \varphi')\]
Invariants

Checking those instructions that assign values to variables (they modify a content of memory):

- an edge labelled with assignment \( x := E \)

\( \varphi'' \) — a formula obtained from formula \( \varphi' \) by renaming of all free occurrences of variable \( x \) to \( x' \)

It is necessary to check the validity of implication

\[
\left( \varphi \land (x' = E) \right) \rightarrow \varphi''
\]

**Example:** Assignment \( k := 3 \cdot k + i + 1 \): 

\[
(\forall n \in \mathbb{Z})(\forall i \in \mathbb{Z})(\forall k \in \mathbb{Z})(\forall k' \in \mathbb{Z}) \left( \varphi \land (k' = 3 \cdot k + i + 1) \rightarrow \varphi'' \right)
\]
Invariants

Finishing the checking that the algorithm for finding maximal element in an array returns a correct result (under assumption that it halts):

- $\psi_0: \varphi_0$
- $\psi_1: \varphi_1 \land (\forall j \in \mathbb{N})(0 \leq j < 1 \rightarrow A[j] \leq A[k])$
- $\psi_2: \varphi_2 \land (\forall j \in \mathbb{N})(0 \leq j < i \rightarrow A[j] \leq A[k])$
- $\psi_3: \varphi_3 \land (\forall j \in \mathbb{N})(0 \leq j < i \rightarrow A[j] \leq A[k])$
- $\psi_4: \varphi_4 \land (\forall j \in \mathbb{N})(0 \leq j < i \rightarrow A[j] \leq A[k]) \land (A[i] > A[k])$
- $\psi_5: \varphi_5 \land (\forall j \in \mathbb{N})(0 \leq j \leq i \rightarrow A[j] \leq A[k])$
- $\psi_6: \varphi_6 \land (\forall j \in \mathbb{N})(0 \leq j < n \rightarrow A[j] \leq A[k])$
- $\psi_7: \varphi_7 \land (result = A[k]) \land (\forall j \in \mathbb{N})(0 \leq j < n \rightarrow A[j] \leq result) \land (\exists j \in \mathbb{N})(0 \leq j < n \land A[j] = result)$
Invariants

Usually it is not necessary to specify invariants in all control states but only in some “important” states — in particular, in states where the algorithm enters or leaves loops:

It is necessary to verify:

- That the invariant holds before entering the loop.
- That if the invariant holds before an iteration of the loop then it holds also after the iteration.
- That the invariant holds when the loop is left.
Invariants

**Example:** In algorithm **Find-Max**, state 2 is such “important” state.

In state 2, the following holds:

- \( n \geq 1 \)
- \( 1 \leq i \leq n \)
- \( 0 \leq k < i \)
- For each \( j \) such that \( 0 \leq j < i \) it holds that \( A[j] \leq A[k] \).
Finiteness of a Computation

Two possibilities how an infinite computation can look:

- some configuration is repeated — then all following configurations are also repeated

- all configurations in a computation are different but a final configuration is never reached
Finiteness of a Computation

One of standard ways of proving that an algorithm halts for every input after a finite number of steps:

- to assign a value from a set \( W \) to every (reachable) configuration
- to define an order \( \leq \) on set \( W \) such that there are no infinite (strictly) decreasing sequences of elements of \( W \)
- to show that the values assigned to configuration decrease with every execution of each instruction, i.e., if \( \alpha \xrightarrow{I} \alpha' \) then
  \[ f(\alpha) > f(\alpha') \]
  (\( f(\alpha), f(\alpha') \) are values from set \( W \) assigned to configurations \( \alpha \) and \( \alpha' \))
Finiteness of a Computation

As a set $W$, we can use for example:

- The set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, \ldots \}$ with ordering $\leq$.

- The set of vectors of natural numbers with lexicographic ordering, i.e., the ordering where vector $(a_1, a_2, \ldots, a_m)$ is smaller than $(b_1, b_2, \ldots, b_n)$, if
  - there exists $i$ such that $1 \leq i \leq m$ and $i \leq n$, where $a_i < b_i$ and for all $j$ such that $1 \leq j < i$ it holds that $a_j = b_j$, or
  - $m < n$ and for all $j$ such that $1 \leq j \leq m$ is $a_j = b_j$.

For example, $(5, 1, 3, 6, 4) < (5, 1, 4, 1)$ and $(4, 1, 1) < (4, 1, 1, 3)$.

**Remark:** The number of elemets in vectors must be bounded by some constant.
Finiteness of a Computation

\[ k := 0 \]
\[ i := 1 \]
\[ k := i \]
\[ i := i + 1 \]

\[ A[i] > A[k] \]
\[ A[i] \leq A[k] \]
\[ i \geq n \]
\[ i < n \]

result := A[k]
Finiteness of a Computation

Example: Vectors assigned to individual configurations:

- State 0: $f(\alpha) = (4)$
- State 1: $f(\alpha) = (3)$
- State 2: $f(\alpha) = (2, n - i, 3)$
- State 3: $f(\alpha) = (2, n - i, 2)$
- State 4: $f(\alpha) = (2, n - i, 1)$
- State 5: $f(\alpha) = (2, n - i, 0)$
- State 6: $f(\alpha) = (1)$
- State 7: $f(\alpha) = (0)$
Computational Complexity of Algorithms
Computers work fast but not infinitely fast. Execution of each instruction takes some (very short) time.

The same problem can be solved by several different algorithms. The time of a computation (determined mostly by the number of executed instructions) can be different for different algorithms.

We can implement the algorithms and then measure the time of their computation. By this we find out how long the computation takes on particular data on which we test the algorithm.

We would like to have a more precise idea how long the computation takes on all possible input data.
Complexity of an Algorithm

- A running time is affected by many factors, e.g.:
  - the algorithm that is used
  - the amount of input data
  - used hardware (e.g., the frequency at which a CPU is running can be important)
  - the used programming language — its implementation (compiler/interpreter)
  - ...

- If we need to solve problem for “small” input data, the running time is usually negligible.

- With increasing amount of input data (the size of input), the running time can grow, sometimes significantly.
Complexity of an Algorithm

- **Time complexity of an algorithm** — how the running time of the algorithm depends on the amount of input data

- **Space complexity of an algorithm** — how the amount of a memory used during a computation grows with respect to the size of input

**Remark:** The precise definitions will be given later.

**Remark:**
- There are also other types of computational complexity, which we will not discuss here (e.g., communication complexity).
Complexity of an Algorithm

To determine the precise running time or the precise amount of used memory just by an analysis of an algorithm can be extremely difficult.

Usually the analysis of complexity of an algorithm involves many simplifications:

- It is usually not analysed how the running time or the amount of used memory depends precisely on particular input data but how they depend on the size of the input.

- Functions expressing how the running time or the amount of used memory grows depending on the size of the input are not computed precisely — instead estimations of these functions are computed.

- Estimations of these functions are usually expressed using asymptotic notation — e.g., it can be said that the running time of MergeSort is $O(n \log n)$, and that the running time of BubbleSort is $O(n^2)$.
Size of the input — a value describing how “big” is an input instance

- In most cases, the size of an input is just one number — it is usually denoted $n$ or $N$.

- Sometimes it is more appropriate to express the size of an input by pair (sometimes even with three, four, etc.) of parameters — in this case, they are oftend denoted $n$ and $m$ (or $N$ and $M$).

- We can choose what should be considered as the size of an input.
Examples, what the size of an input can be:

- An input is a sequence of some values, an array of elements, etc. (e.g., in problems like sorting, searching in an array, finding the maximal element, etc.):
  \[ n \quad \text{— the number of elements in this sequence or array} \]

- An input is a string of characters (a word from some alphabet):
  \[ n \quad \text{— the number of characters in this string} \]

- An input consists of two strings, e.g., a (long) text that will be searched through, and a (shorter) searched pattern:
  \[ n \quad \text{— the number of characters in the text} \]
  \[ m \quad \text{— the number of characters in the searched pattern} \]
An input is a set of strings:

One possibility:

\[ n \] — the sum of lengths of all strings

Other variant:

\[ n \] — the sum of lengths of all strings, \[ m \] — the number of strings

The input is a graph:

\[ n \] — the number of nodes, \[ m \] — the number of edges
The input is one number (e.g., in the primarity testing):

One possibility:

- $n$ — the number of bits of the number — e.g., the size of input 962261 is 20

Other variant:

- $n$ — the value of the number — the size of input 962261 is 962261

The input is a sequence of numbers, and the running time is affected by the values of the numbers (e.g., in the problem where the goal is to compute the greatest common divisor of all numbers in a given sequence):

- $n$ — the sum of numbers of bits of all numbers in the given sequence
Running Time

Let us say that we have:

- an algorithm $Alg$ solving a problem $P$ (resp. a particular implementation of algorithm $Alg$),
- a machine $M$ executing the algorithm $Alg$,
- an input $w$ from the set $In$, which is a set of all inputs of problem $P$

An example:

- a particular implementation of Quicksort in C++ solving the problem of sorting,
- a computer with some particular type of processor working on some particular frequency, with some particular amount of memory, operating system, etc.
- input: array $[6, 13, 1, 8, 4, 5, 8]$
  (remark: a more realistic example would be an array with one million elements)
Running Time

\( t(w) \) — the running time of the algorithm \( Alg \) on input \( w \) on machine \( M \)

What units should be used for expressing time? (As we will see, this is not important when asymptotic notation is used.)

- **in seconds** — it depends on too many details of implementation, it is difficult to determine it in other way than by measurement (even on the same computer with the same data the running time can fluctuate)

- **the number of steps** — it must be specified what is considered as one step, for example:
  - one statement of a high level programming language
  - one instruction of machine code or bytecode
  - one tick of a processor
  - one operation of some particular type — e.g., a comparison, an arithmetic operation, etc. (while all other operations are ignored)
  - ...
Let us say that an algorithm is represented by a control-flow graph:

- To every instruction (i.e., to every edge) we assign a value specifying how long it takes to perform this instruction once.
- The execution time of different instructions can be different.
- For simplicity we assume that an execution of the same instruction takes always the same time — the value assigned to an instruction is a number from the set \( \mathbb{R}^+ \) (the set of nonnegative real numbers).
Running Time

\[ k := 0 \]

\[ i := 1 \]

\[ i \geq n \]

\[ [i < n] \]

\[ [i \geq n] \]

\[ A[i] > A[k] \]

\[ i := i + 1 \]

\[ [A[i] \leq A[k]] \]

\[ [A[i] > A[k]] \]

\[ k := i \]

\[ result := A[k] \]

---

**Instr.** | **time**
---|---
\[ k := 0 \] | \( c_0 \)
\[ i := 1 \] | \( c_1 \)
\[ [i < n] \] | \( c_2 \)
\[ [i \geq n] \] | \( c_3 \)
\[ [A[i] \leq A[k]] \] | \( c_4 \)
\[ [A[i] > A[k]] \] | \( c_5 \)
\[ k := i \] | \( c_6 \)
\[ i := i + 1 \] | \( c_7 \)
\[ result := A[k] \] | \( c_8 \)
**Example:** The execution times of individual instructions could be for example:

<table>
<thead>
<tr>
<th>Instr.</th>
<th>symbol</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k := 0$</td>
<td>$c_0$</td>
<td>4</td>
</tr>
<tr>
<td>$i := 1$</td>
<td>$c_1$</td>
<td>4</td>
</tr>
<tr>
<td>$[i &lt; n]$</td>
<td>$c_2$</td>
<td>10</td>
</tr>
<tr>
<td>$[i \geq n]$</td>
<td>$c_3$</td>
<td>12</td>
</tr>
<tr>
<td>$A[i] \leq A[k]$</td>
<td>$c_4$</td>
<td>14</td>
</tr>
<tr>
<td>$A[i] &gt; A[k]$</td>
<td>$c_5$</td>
<td>12</td>
</tr>
<tr>
<td>$k := i$</td>
<td>$c_6$</td>
<td>5</td>
</tr>
<tr>
<td>$i := i + 1$</td>
<td>$c_7$</td>
<td>6</td>
</tr>
<tr>
<td>$result := A[k]$</td>
<td>$c_8$</td>
<td>5</td>
</tr>
</tbody>
</table>

For a particular input $w$, e.g., for $w = ([3, 8, 4, 5, 2], 5)$, we could simulate the computation and determine the precise running time $t(w)$. 
Time Complexity of an Algorithm

Let us say that:

- For a given algorithm $Alg$ and machine $M$, and for every input $w$ from the set of all inputs $In$, the running time $t(w)$ is precisely defined.

- To each input $w$ from set $In$, a number $size(w)$ describing the size of the input $w$ is assigned.

(Formally, it is a function $size : In \rightarrow \mathbb{N}$.)

**Definition**

The **time complexity of algorithm $Alg$ in the worst case** is the function $T : \mathbb{N} \rightarrow \mathbb{R}^+$ that assigns to each natural number $n$ the maximal running time of the algorithm $Alg$ on an input of size $n$.

So for each $n \in \mathbb{N}$ we have:

- For each input $w \in In$ such that $size(w) = n$ is $t(w) \leq T(n)$.

- There exists an input $w \in In$ such that $size(w) = n$ and $t(w) = T(n)$. 
It is obvious from this definition that the time complexity of an algorithm is a function whose precise values depend not only on the given algorithm $Alg$ but also on the following things:

- on a machine $M$, on which the algorithm $Alg$ runs,
- on the precise definition of the running time $t(w)$ of algorithm $Alg$ on machine $M$ with input $w \in In$,
- on the precise definition of the size of an input (i.e., on the definition of function $size$).
Sometimes, the time complexity in the **average case** is also analyzed:

- Some particular **probabilistic distribution** on the set of inputs must be assumed.
- Instead of the maximal running time on inputs of size $n$, the expected value of the running times is considered.
- Usually, the analysis of the average case is much more complicated than the analysis of the worst case.
- Often, these two functions are not very different but sometimes the difference is significant.

**Remark:** It usually makes little sense to analyze the time complexity in the best case.
An example of an analysis of the time complexity of algorithm **Find-Max without** the use of asymptotic notation:

- Such precise analysis is almost never done in practice — it is too tedious and complicated.

- This illustrates what things are ignored in an analysis where asymptotic notation is used and how much the analysis is simplified by this.

- We will compute with constants $c_0, c_1, \ldots, c_8$, which specify the execution time of individual instructions — we won’t compute with concrete numbers.
The inputs are of the form \((A, n)\), where \(A\) is an array and \(n\) is the number of elements in this array (where \(n \geq 1\)).

We take \(n\) as the size of input \((A, n)\).

Consider now some particular input \(w = (A, n)\) of size \(n\):

- The running time \(t(w)\) on input \(w\) can be expressed as
  \[
  t(w) = c_0 m_0 + c_1 m_1 + \cdots + c_8 m_8,
  \]
  where \(m_0, m_1, \ldots, m_8\) are numbers specifying how many times is each instruction performed in the computation on input \(w\).
### Time Complexity of an Algorithm

<table>
<thead>
<tr>
<th>Instr.</th>
<th>time</th>
<th>occurences</th>
<th>value of $m_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k := 0$</td>
<td>$c_0$</td>
<td>$m_0$</td>
<td>1</td>
</tr>
<tr>
<td>$i := 1$</td>
<td>$c_1$</td>
<td>$m_1$</td>
<td>1</td>
</tr>
<tr>
<td>$[i &lt; n]$</td>
<td>$c_2$</td>
<td>$m_2$</td>
<td>1</td>
</tr>
<tr>
<td>$[i \geq n]$</td>
<td>$c_3$</td>
<td>$m_3$</td>
<td>$n - 1$</td>
</tr>
<tr>
<td>$[A[i] \leq A[k]]$</td>
<td>$c_4$</td>
<td>$m_4$</td>
<td>1</td>
</tr>
<tr>
<td>$[A[i] &gt; A[k]]$</td>
<td>$c_5$</td>
<td>$m_5$</td>
<td>$n - 1 - \ell$</td>
</tr>
<tr>
<td>$k := i$</td>
<td>$c_6$</td>
<td>$m_6$</td>
<td>$\ell$</td>
</tr>
<tr>
<td>$i := i + 1$</td>
<td>$c_7$</td>
<td>$m_7$</td>
<td>$\ell$</td>
</tr>
<tr>
<td>$\text{result} := A[k]$</td>
<td>$c_8$</td>
<td>$m_8$</td>
<td>$n - 1$</td>
</tr>
</tbody>
</table>

\(\ell\) — the number of iterations of the cycle where \(A[i] > A[k]\)  
(obviously \(0 \leq \ell < n\))
By assigning values to

\[ t(w) = c_0m_0 + c_1m_1 + \cdots + c_8m_8, \]

we obtain

\[ t(w) = d_1 + d_2 \cdot (n-1) + d_3 \cdot (n-1 - \ell) + d_4 \cdot \ell, \]

where

\[
\begin{align*}
  d_1 &= c_0 + c_1 + c_3 + c_8 \\
  d_2 &= c_2 + c_7 \\
  d_3 &= c_4 \\
  d_4 &= c_5 + c_6
\end{align*}
\]

After simplification we have

\[ t(w) = (d_2 + d_3) \cdot n + (d_4 - d_3) \cdot \ell + (d_1 - d_2 - d_3) \]

**Remark:** \( t(w) \) is not the time complexity but the running time for a particular input \( w \)
For example, if the execution times of instructions will be:

<table>
<thead>
<tr>
<th>Instr.</th>
<th>symb.</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k := 0$</td>
<td>$c_0$</td>
<td>4</td>
</tr>
<tr>
<td>$i := 1$</td>
<td>$c_1$</td>
<td>4</td>
</tr>
<tr>
<td>$[i &lt; n]$</td>
<td>$c_2$</td>
<td>10</td>
</tr>
<tr>
<td>$[i \geq n]$</td>
<td>$c_3$</td>
<td>12</td>
</tr>
<tr>
<td>$[A[i] \leq A[k]]$</td>
<td>$c_4$</td>
<td>14</td>
</tr>
<tr>
<td>$[A[i] &gt; A[k]]$</td>
<td>$c_5$</td>
<td>12</td>
</tr>
<tr>
<td>$k := i$</td>
<td>$c_6$</td>
<td>5</td>
</tr>
<tr>
<td>$i := i + 1$</td>
<td>$c_7$</td>
<td>6</td>
</tr>
<tr>
<td>result := $A[k]$</td>
<td>$c_8$</td>
<td>5</td>
</tr>
</tbody>
</table>

then $d_1 = 25$, $d_2 = 16$, $d_3 = 14$, and $d_4 = 17$.

In this case is $t(w) = 30n + 3\ell - 5$.

For the input $w = ([3, 8, 4, 5, 2], 5)$ is $n = 5$ and $\ell = 1$, therefore $t(w) = 30 \cdot 5 + 3 \cdot 1 - 5 = 148$. 
It can depend on details of implementation and on the precise values of constants, for which inputs of size $n$ the computation takes the longest time (i.e., which are the worst cases):

The running time of algorithm $\text{FIND-MAX}$ for an input $w = (A, n)$ of size $n$:

$$t(w) = (d_2 + d_3) \cdot n + (d_4 - d_3) \cdot \ell + (d_1 - d_2 - d_3)$$

- If $d_3 \geq d_4$ — the worst cases are those where $\ell$ has the smallest value $\ell = 0$ — for example inputs of the form $[0, 0, \ldots, 0]$ or of the form $[n, n-1, n-2, \ldots, 2, 1]$

- If $d_3 \leq d_4$ — the worst are those cases where $\ell$ has the greatest value $\ell = n-1$ — for example inputs of the form $[0, 1, \ldots, n-1]$
The time complexity $T(n)$ of algorithm $\text{Find-Max}$ in the worst case is given as follows:

- If $d_3 \geq d_4$:
  \[ T(n) = (d_2 + d_3) \cdot n + (d_1 - d_2 - d_3) \]

- If $d_3 \leq d_4$:
  \[ T(n) = (d_2 + d_3) \cdot n + (d_4 - d_3) \cdot (n - 1) + (d_1 - d_2 - d_3) \]
  \[ = (d_2 + d_4) \cdot n + (d_1 - d_2 - d_4) \]

**Example:** For $d_1 = 25$, $d_2 = 16$, $d_3 = 14$, $d_4 = 17$ is

\[ T(n) = (16 + 17) \cdot n + (25 - 16 - 17) \]
\[ = 33n - 8 \]
In both cases (when $d_3 \geq d_4$ or when $d_3 \leq d_4$), the time complexity of the algorithm $\text{Find-Max}$ is a function

$$T(n) = an + b$$

where $a$ and $b$ are some constants whose precise values depend on the execution time of individual instructions.

**Remark:** These constants could be expressed as

$$a = d_2 + \max\{d_3, d_4\} \quad b = d_1 - d_2 - \max\{d_3, d_4\}$$

For example

$$T(n) = 33n - 8$$
If it would be sufficient to find out that the time complexity of the algorithm `Find-Max` is some function of the form

\[ T(n) = an + b, \]

where the precise values of constants \( a \) and \( b \) would not be important for us, the whole analysis could be considerably simpler.

- In fact, we usually do not want to know precisely how function \( T(n) \) look (in general, it can be a very complicated function), and it would be sufficient to know that values of the function \( T(n) \) “approximately” correspond to values of a function \( S(n) = an + b \), where \( a \) and \( b \) are some constants.
For a given function $T(n)$ expressing the time or space complexity, it is usually sufficient to express it approximately — to have an estimation where

- we ignore the less important parts
  (e.g., in function $T(n) = 15n^2 + 40n - 5$ we can ignore $40n$ and $-5$, and to consider function $T(n) = 15n^2$ instead of the original function),

- we ignore multiplication constants
  (e.g., instead of function $T(n) = 15n^2$ we will consider function $T(n) = n^2$)

- we won’t ignore constants in exponents — for example there is a big difference between functions $T_1(n) = n^2$ and $T_2(n) = n^3$.

- we will be interested how function $T(n)$ behaves for “big” values of $n$, we can ignore its behaviour on small values
A program works on an input of size $n$. Let us assume that for an input of size $n$, the program performs $T(n)$ operations and that an execution of one operation takes 1 $\mu$s ($10^{-6}$ s).

<table>
<thead>
<tr>
<th>$n$</th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>20 $\mu$s</td>
<td>40 $\mu$s</td>
<td>60 $\mu$s</td>
<td>80 $\mu$s</td>
<td>0.1 ms</td>
<td>0.2 ms</td>
<td>0.5 ms</td>
<td>1 ms</td>
</tr>
<tr>
<td>$n \log n$</td>
<td>86 $\mu$s</td>
<td>0.213 ms</td>
<td>0.354 ms</td>
<td>0.506 ms</td>
<td>0.664 ms</td>
<td>1.528 ms</td>
<td>4.48 ms</td>
<td>9.96 ms</td>
</tr>
<tr>
<td>$n^2$</td>
<td>0.4 ms</td>
<td>1.6 ms</td>
<td>3.6 ms</td>
<td>6.4 ms</td>
<td>10 ms</td>
<td>40 ms</td>
<td>0.25 s</td>
<td>1 s</td>
</tr>
<tr>
<td>$n^3$</td>
<td>8 ms</td>
<td>64 ms</td>
<td>0.216 s</td>
<td>0.512 s</td>
<td>1 s</td>
<td>8 s</td>
<td>125 s</td>
<td>16.7 min.</td>
</tr>
<tr>
<td>$n^4$</td>
<td>0.16 s</td>
<td>2.56 s</td>
<td>12.96 s</td>
<td>42 s</td>
<td>100 s</td>
<td>26.6 min.</td>
<td>17.36 hours</td>
<td>11.57 days</td>
</tr>
<tr>
<td>$2^n$</td>
<td>1.05 s</td>
<td>12.75 days</td>
<td>36560 years</td>
<td>$38.3 \cdot 10^9$ years</td>
<td>$40.1 \cdot 10^{15}$ years</td>
<td>$50 \cdot 10^{45}$ years</td>
<td>$10.4 \cdot 10^{136}$ years</td>
<td>–</td>
</tr>
<tr>
<td>$n!$</td>
<td>77147 years</td>
<td>$2.59 \cdot 10^{34}$ years</td>
<td>$2.64 \cdot 10^{68}$ years</td>
<td>$2.27 \cdot 10^{105}$ years</td>
<td>$2.96 \cdot 10^{144}$ years</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Let us consider 3 algorithms with complexities 
\( T_1(n) = n, \ T_2(n) = n^3, \ T_3(n) = 2^n \). Our computer can do in a reasonable time (the time we are willing to wait) \( 10^{12} \) steps.

\[
\begin{array}{|c|c|}
\hline
\text{Complexity} & \text{Input size} \\
\hline
T_1(n) = n & 10^{12} \\
T_2(n) = n^3 & 10^4 \\
T_3(n) = 2^n & 40 \\
\hline
\end{array}
\]
Growth of Functions

Let us consider 3 algorithms with complexities 
\[ T_1(n) = n, \quad T_2(n) = n^3, \quad T_3(n) = 2^n. \] 
Our computer can do in a reasonable time (the time we are willing to wait) \(10^{12}\) steps.

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Input size</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_1(n) = n )</td>
<td>(10^{12})</td>
</tr>
<tr>
<td>( T_2(n) = n^3 )</td>
<td>(10^4)</td>
</tr>
<tr>
<td>( T_3(n) = 2^n )</td>
<td>40</td>
</tr>
</tbody>
</table>

Now we speed up our computer 1000 times, meaning it can do \(10^{15}\) steps.

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Input size</th>
<th>Growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_1(n) = n )</td>
<td>(10^{15})</td>
<td>(1000\times)</td>
</tr>
<tr>
<td>( T_2(n) = n^3 )</td>
<td>(10^5)</td>
<td>(10\times)</td>
</tr>
<tr>
<td>( T_3(n) = 2^n )</td>
<td>50</td>
<td>(+10)</td>
</tr>
</tbody>
</table>
Asymptotic Notation

In the following, we will consider functions of the form \( f : \mathbb{N} \rightarrow \mathbb{R} \), where:

- The values of \( f(n) \) need not to be defined for all values of \( n \in \mathbb{N} \) but there must exist some constant \( n_0 \) such that the value of \( f(n) \) is defined for all \( n \in \mathbb{N} \) such that \( n \geq n_0 \).

**Example:** Function \( f(n) = \log_2(n) \) is not defined for \( n = 0 \) but it is defined for all \( n \geq 1 \).

- There must exist a constant \( n_0 \) such that for all \( n \in \mathbb{N} \), where \( n \geq n_0 \), is \( f(n) \geq 0 \).

**Example:** It holds for function \( f(n) = n^2 - 25 \) that \( f(n) \geq 0 \) for all \( n \geq 5 \).
Asymptotic Notation

Let us take an arbitrary function \( f : \mathbb{N} \rightarrow \mathbb{R} \). Expressions \( O(f) \), \( \Omega(f) \), and \( \Theta(f) \) denote sets of functions of the type \( \mathbb{N} \rightarrow \mathbb{R} \), where:

- \( O(f) \) – the set of all functions that grow at most as fast as \( f \)
- \( \Omega(f) \) – the set of all functions that grow at least as fast as \( f \)
- \( \Theta(f) \) – the set of all functions that grow as fast as \( f \)

**Remark:** These are not definitions! The definitions will follow on the next slides.

- \( O \) – big “\( O \)”
- \( \Omega \) – uppercase Greek letter “omega”
- \( \Theta \) – uppercase Greek letter “theta”
Asymptotic Notation – Symbol $O$

**Definition**

Let us consider an arbitrary function $g : \mathbb{N} \to \mathbb{R}$. For a function $f : \mathbb{N} \to \mathbb{R}$ we have $f \in O(g)$ iff

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)(f(n) \leq c g(n)).$$
Remarks:

- \( c \) is a positive real number (i.e., \( c \in \mathbb{R} \) and \( c > 0 \))
- \( n_0 \) and \( n \) are natural numbers (i.e., \( n_0 \in \mathbb{N} \) and \( n \in \mathbb{N} \))
**Example:** Let us consider functions \( f(n) = 2n^2 + 3n + 7 \) and \( g(n) = n^2 \).

We want to show that \( f \in O(g) \), i.e., \( f \in O(n^2) \):

- **Approach 1:**
  
  Let us take for example \( c = 3 \).

  \[
  cg(n) = 3n^2 = 2n^2 + \frac{1}{2}n^2 + \frac{1}{2}n^2
  \]

  We need to find some \( n_0 \) such that for all \( n \geq n_0 \) it holds that

  \[
  2n^2 \geq 2n^2 \quad \frac{1}{2}n^2 \geq 3n \quad \frac{1}{2}n^2 \geq 7
  \]

  We can easily check that for example \( n_0 = 6 \) satisfies this.

  For each \( n \geq 6 \) we have \( cg(n) \geq f(n) \):

  \[
  cg(n) = 3n^2 = 2n^2 + \frac{1}{2}n^2 + \frac{1}{2}n^2 \geq 2n^2 + 3n + 7 = f(n)
  \]
Asymptotic Notation – Symbol $O$

The example where $f(n) = 2n^2 + 3n + 7$ and $g(n) = n^2$:

- **Approach 2:**
  
  Let us take $c = 12$.

  $$cg(n) = 12n^2 = 2n^2 + 3n^2 + 7n^2$$

  We need to find some $n_0$ such that for all $n \geq n_0$ we have
  
  $$2n^2 \geq 2n^2$$  
  $$3n^2 \geq 3n$$  
  $$7n^2 \geq 7$$

  These inequalities obviously hold for $n_0 = 1$, and so for each $n \geq 1$ we have $cg(n) \geq f(n)$:

  $$cg(n) = 12n^2 = 2n^2 + 3n^2 + 7n^2 \geq 2n^2 + 3n + 7 = f(n)$$
Asymptotic Notation – Symbol $O$

### Proposition

Let us assume that $a$ and $b$ are constants such that $a > 0$ and $b > 0$, and $k$ and $\ell$ are some arbitrary constants where $k \geq 0$, $\ell \geq 0$ and $k < \ell$.

Let us consider functions

\[
  f(n) = a \cdot n^k \quad \text{and} \quad g(n) = b \cdot n^\ell
\]

For each such functions $f$ and $g$ it holds that $f \in O(g)$:

**Proof:** Let us take $c = \frac{a}{b}$.

Because for $n \geq 1$ we obviously have $n^k \leq n^\ell$ (since $k \leq \ell$), for $n \geq 1$ we have

\[
  c \cdot g(n) = \frac{a}{b} \cdot g(n) = \frac{a}{b} \cdot b \cdot n^\ell = a \cdot n^\ell \geq a \cdot n^k = f(n)
\]

Z. Sawa (TU Ostrava)  Theoretical Computer Science  September 13, 2019  368 / 490
Asymptotic Notation – Symbol $\Omega$

**Definition**

Let us consider an arbitrary function $g : \mathbb{N} \rightarrow \mathbb{R}$. For a function $f : \mathbb{N} \rightarrow \mathbb{R}$ we have $f \in \Omega(g)$ iff

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0) \left( c g(n) \leq f(n) \right).$$
It is not difficult to prove the following proposition:

For arbitrary functions $f$ and $g$ we have:

$$f \in O(g) \text{ iff } g \in \Omega(f)$$
Definition

Let us consider an arbitrary function $g : \mathbb{N} \to \mathbb{R}$. For a function $f : \mathbb{N} \to \mathbb{R}$ we have $f \in \Theta(g)$ iff

$$(\exists c_1 > 0)(\exists c_2 > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)\left( c_1 g(n) \leq f(n) \leq c_2 g(n) \right).$$
For arbitrary functions $f$ and $g$ we have:

\[ f \in \Theta(g) \quad \text{iff} \quad f \in O(g) \text{ and } f \in \Omega(g) \]
\[ f \in \Theta(g) \quad \text{iff} \quad f \in O(g) \text{ and } g \in O(f) \]
\[ f \in \Theta(g) \quad \text{iff} \quad g \in \Theta(f) \]
Asymptotic Notation

For arbitrary functions $f$, $g$, and $h$ we have:

- if $f \in O(g)$ and $g \in O(h)$ then $f \in O(h)$
- if $f \in \Omega(g)$ and $g \in \Omega(h)$ then $f \in \Omega(h)$
- if $f \in \Theta(g)$ and $g \in \Theta(h)$ then $f \in \Theta(h)$
Examples:

\[ n \in O(n^2) \]
\[ 1000n \in O(n) \]
\[ 2^{\log_2 n} \in \Theta(n) \]
\[ n^3 \not\in O(n^2) \]
\[ n^2 \not\in O(n) \]
\[ n^3 + 2^n \not\in O(n^2) \]

\[ n^3 \in O(n^4) \]
\[ 0.00001n^2 - 10^{10}n \in \Theta(10^{10}n^2) \]
\[ n^3 - n^2 \log_2 n + 1000n - 10^{100} \in \Theta(n^3) \]
\[ n^3 + 1000n - 10^{100} \in O(n^3) \]
\[ n^3 + n^2 \not\in \Theta(n^2) \]
\[ n! \not\in O(2^n) \]
There are pairs of functions $f$ and $g$ such that
\[ f \not\in O(g) \quad \text{and} \quad g \not\in O(f), \]
for example
\[ f(n) = n \quad \text{and} \quad g(n) = n^{1+\sin(n)}. \]

$O(1)$ is the set of all bounded functions, i.e., functions whose function values can be bounded from above by a constant.
Asymptotic Notation

For any pair of functions \( f, g \) we have:

- \( \max(f, g) \in \Theta(f + g) \)
- if \( f \in O(g) \) then \( f + g \in \Theta(g) \)

For any functions \( f_1, f_2, g_1, g_2 \) we have:

- if \( f_1 \in O(f_2) \) and \( g_1 \in O(g_2) \) then \( f_1 + g_1 \in O(f_2 + g_2) \) and \( f_1 \cdot g_1 \in O(f_2 \cdot g_2) \)
- if \( f_1 \in \Theta(f_2) \) and \( g_1 \in \Theta(g_2) \) then \( f_1 + g_1 \in \Theta(f_2 + g_2) \) and \( f_1 \cdot g_1 \in \Theta(f_2 \cdot g_2) \)
A function $f$ is called:

- **logarithmic**, if $f(n) \in \Theta(\log n)$
- **linear**, if $f(n) \in \Theta(n)$
- **quadratic**, if $f(n) \in \Theta(n^2)$
- **cubic**, if $f(n) \in \Theta(n^3)$
- **polynomial**, if $f(n) \in O(n^k)$ for some $k > 0$
- **exponential**, if $f(n) \in O(c^{nk})$ for some $c > 1$ and $k > 0$

Exponential functions are often written in the form $2^{O(n^k)}$ when the asymptotic notation is used, since then we do not need to consider different bases.
Asymptotic Notation

As mentioned before, expressions $O(g)$, $\Omega(g)$, and $\Theta(g)$ denote certain sets of functions.

In some texts, these expressions are sometimes used with a slightly different meaning:

- an expression $O(g)$, $\Omega(g)$ or $\Theta(g)$ does not represent the corresponding set of functions but some function from this set.

This convention is often used in equations and inequations.

**Example:** $3n^3 + 5n^2 - 11n + 2 = 3n^3 + O(n^2)$

When using this convention, we can for example write $f = O(g)$ instead of $f \in O(g)$. 
Complexity of Algorithms

Let us say we would like to analyze the time complexity $T(n)$ of some algorithm consisting of instructions $I_1, I_2, \ldots, I_k$:

- If $m_1, m_2, \ldots, m_k$ are the numbers of executions of individual instructions for some input $w$ (i.e., the instruction $I_i$ is performed $m_i$ times for the input $w$), then the total number of executed instructions for input $w$ is
  \[ T(n) = c_1 m_1 + c_2 m_2 + \cdots + c_k m_k. \]

- Let us consider functions $f_1, f_2, \ldots, f_k$, where $f_i : \mathbb{N} \to \mathbb{R}$, and where $f_i(n)$ is the maximum of numbers of executions of instruction $I_i$ for all inputs of size $n$.

- Obviously, $T \in \Omega(f_i)$ for any function $f_i$.

- It is also obvious that $T \in O(f_1 + f_2 + \cdots + f_k)$. 
Let us recall that if \( f \in O(g) \) then \( f + g \in O(g) \).

If there is a function \( f_i \) such that for all \( f_j \), where \( j \neq i \), we have \( f_j \in O(f_i) \), then

\[
T \in O(f_i)\text{.}
\]

This means that in an analysis of the time complexity \( T(n) \), we can restrict our attention to the number of executions of the instruction that is performed most frequently (and which is performed at most \( f_i(n) \) times for an input of size \( n \)), since we have

\[
T \in \Theta(f_i)\text{.}
\]
**Example:** In the analysis of the complexity of the searching of a number in a sequence we obtained

\[
f(n) = an + b.
\]

If we would not like to do such a detailed analysis, we could deduce that the time complexity of the algorithm is \(\Theta(n)\), because:

- The algorithm contains only one cycle, which is performed \((n - 1)\) times for an input of size \(n\), the number of iterations of the cycle is in \(\Theta(n)\).

- Several instructions are performed in one iteration of the cycle. The number of these instructions is bounded from both above and below by some constant independent on the size of the input.

- Other instructions are performed at most once, and so they contribute to the total running time by adding a constant.
Let us try to analyze the time complexity of the following algorithm:

**Algorithm 2: Insertion sort**

1. **INSERTION-SORT** \((A, n)\):
2. begin
3. \(\text{for } j := 1 \text{ to } n-1 \text{ do} \)
4. \(x := A[j]\)
5. \(i := j - 1\)
6. \(\text{while } i \geq 0 \text{ and } A[i] > x \text{ do} \)
7. \(A[i + 1] := A[i]\)
8. \(i := i - 1\)
9. end
10. \(A[i + 1] := x\)
11. end
12. end

I.e., we want to find a function \(T(n)\) such that the time complexity of the algorithm **INSERTION-SORT** in the worst case is in \(\Theta(T(n))\).
Let us consider inputs of size $n$:

- The outer cycle `for` is performed at most $n - 1$ times.
- The inner cycle `while` is performed at most $(j - 1)$ times for a given value $j$.
- There are inputs such that the cycle `while` is performed exactly $(j - 1)$ times for each value $j$ from $2$ to $n$.
- So in the worst case, the cycle `while` is performed exactly $m$ times, where
  \[ m = 1 + 2 + \cdots + (n - 1) = (1 + (n - 1)) \cdot \frac{n-1}{2} = \frac{1}{2} n^2 - \frac{1}{2} n \]
- This means that the total running time of the algorithm `Insertion-Sort` in the worst case is $\Theta(n^2)$. 
In the previous case, we have computed the total number of executions of the cycle `while` accurately.

This is not always possible in general, or it can be quite complicated. It is also not necessary, if we only want an asymptotic estimation.
For example, if we were not able to compute the sum of the arithmetic progression, we could proceed as follows:

- The outer cycle `for` is not performed more than \( n \) times and the inner cycle `while` is performed at most \( n \) times in each iteration of the outer cycle.

So we have \( T \in O(n^2) \).

- For some inputs, the cycle `while` is performed at least \( \lceil n/2 \rceil \) times in the last \( \lfloor n/2 \rfloor \) iterations of the cycle `for`.

So the cycle `while` is performed at least \( \lceil n/2 \rceil \cdot \lfloor n/2 \rfloor \) times for some inputs.

\[
\lceil n/2 \rceil \cdot \lfloor n/2 \rfloor \geq (n/2 - 1) \cdot (n/2) = \frac{1}{4} n^2 - \frac{1}{2} n
\]

This implies \( T \in \Omega(n^2) \).
NP-Complete Problems
There is a **polynomial reduction** of problem $P_1$ to problem $P_2$ if there exists an algorithm $Alg$ with a polynomial time complexity that reduces problem $P_1$ to problem $P_2$. 
Polynomial Reductions between Problems

Inputs of problem $P_1$

Inputs of problem $P_2$
Polynomial Reductions between Problems

Inputs of problem $P_1$  

No  

Yes  

Inputs of problem $P_2$  

No  

Yes  

Alg
Let us say that problem $A$ can be reduced in polynomial time to problem $B$, i.e., there is a (polynomial) algorithm $P$ realizing this reduction.

If problem $B$ is in the class $\text{PTIME}$ then problem $A$ is also in the class $\text{PTIME}$.

A solution of problem $A$ for an input $x$:

- Call $P$ with input $x$ and obtain a returned value $P(x)$.
- Call a polynomial time algorithm solving problem $B$ with the input $P(x)$.
  Write the returned value as the answer for $A$.

That means:

If $A$ is not in $\text{PTIME}$ then also $B$ can not be in $\text{PTIME}$. 
There is a big class of algorithmic problems called **NP-complete** problems such that:

- these problems can be solved by exponential time algorithms
- no polynomial time algorithm is known for any of these problems
- on the other hand, for any of these problems it is not proved that there cannot exist a polynomial time algorithm for the given problem
- every NP-complete problem can be polynomially reduced to any other NP-complete problem

**Remark:** This is not a definition of NP-complete problems. The precise definition will be described later.
A typical example of an NP-complete problem is the SAT problem:

**SAT (boolean satisfiability problem)**

- **Input:** Boolean formula $\varphi$.
- **Question:** Is $\varphi$ satisfiable?

**Example:**

Formula $\varphi_1 = x_1 \land (\neg x_2 \lor x_3)$ is satisfiable:

e.g., for valuation $\nu$ where $\nu(x_1) = 1$, $\nu(x_2) = 0$, $\nu(x_3) = 1$, the formula $\varphi_1$ is true.

Formula $\varphi_2 = (x_1 \land \neg x_1) \lor (\neg x_2 \land x_3 \land x_2)$ is not satisfiable:

it is false for every valuation $\nu$. 
3-SAT is a variant of the SAT problem where the possible inputs are restricted to formulas of a certain special form:

<table>
<thead>
<tr>
<th><strong>3-SAT</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Formula $\varphi$ is a conjunctive normal form where every clause contains exactly 3 literals.</td>
</tr>
<tr>
<td><strong>Question:</strong> Is $\varphi$ satisfiable?</td>
</tr>
</tbody>
</table>
Problem 3-SAT

Recalling some notions:

- A **literal** is a formula of the form $x$ or $\neg x$ where $x$ is an atomic proposition.

- A **clause** is a disjunction of literals.

  Examples: $x_1 \lor \neg x_2$ $\neg x_5 \lor x_8 \lor \neg x_{15} \lor \neg x_{23}$ $x_6$

- A formula is in a **conjuctive normal form (CNF)** if it is a conjunction of clauses.

  Example: $(x_1 \lor \neg x_2) \land (\neg x_5 \lor x_8 \lor \neg x_{15} \lor \neg x_{23}) \land x_6$

So in the 3-SAT problem we require that a formula $\varphi$ is in a CNF and moreover that every clause of $\varphi$ contains exactly three literals.

**Example:**

$$(x_1 \lor \neg x_2 \lor x_4) \land (\neg x_1 \lor x_3 \lor x_3) \land (\neg x_1 \lor \neg x_3 \lor \neg x_4) \land (x_2 \lor \neg x_3 \lor x_4)$$
The following formula is satisfiable:
\[(x_1 \lor \neg x_2 \lor x_4) \land (\neg x_1 \lor x_3 \lor x_3) \land (\neg x_1 \lor \neg x_3 \lor \neg x_4) \land (x_2 \lor \neg x_3 \lor x_4)\]

It is true for example for valuation \( \nu \) where
\[
\begin{align*}
\nu(x_1) &= 0 \\
\nu(x_2) &= 1 \\
\nu(x_3) &= 0 \\
\nu(x_4) &= 1
\end{align*}
\]

On the other hand, the following formula is not satisfiable:
\[
(x_1 \lor x_1 \lor x_1) \land (\neg x_1 \lor \neg x_1 \lor \neg x_1)
\]
As an example, a polynomial time reduction from the 3-SAT problem to the independent set problem (IS) will be described.

Remark: Both 3-SAT and IS are examples of NP-complete problems.
Independent Set (IS) Problem

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

**Question:** Is there an independent set of size $k$ in the graph $G$?

**Remark:** An **independent set** in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.

$k = 4$
Independent Set (IS) Problem

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

**Question:** Is there an independent set of size $k$ in the graph $G$?

**Remark:** An **independent set** in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.
Independent Set (IS) Problem

An example of an instance where the answer is **Yes**:

![Graph with nodes and edges indicating an independent set for k = 4]

An example of an instance where the answer is **No**:

![Graph with nodes and edges indicating no independent set for k = 5]
A Reduction from 3-SAT to IS

We describe a (polynomial-time) algorithm with the following properties:

- **Input:** An arbitrary instance of 3-SAT, i.e., a formula $\varphi$ in a conjunctive normal form where every clause contains exactly three literals.

- **Output:** An instance of IS, i.e., an undirected graph $G$ and a number $k$.

Moreover, the following will be ensured for an arbitrary input (i.e., for an arbitrary formula $\varphi$ in the above mentioned form):

There will be an independent set of size $k$ in graph $G$ iff formula $\varphi$ will be satisfiable.
A Reduction from 3-SAT to IS

\((x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4) \land (x_1 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor x_2 \lor x_4)\)
A Reduction from 3-SAT to IS

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4) \land (x_1 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor x_2 \lor x_4)\]

For each occurrence of a literal we add a node to the graph.
A Reduction from 3-SAT to IS

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4) \land (x_1 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor x_2 \lor x_4)\]

We connect with edges the nodes corresponding to occurrences of literals belonging to the same clause.
A Reduction from 3-SAT to IS

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4) \land (x_1 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor x_2 \lor x_4)\]

For each pair of nodes corresponding to literals \(x_i\) and \(\neg x_i\) we add an edge between them.
A Reduction from 3-SAT to IS

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4) \land (x_1 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor x_2 \lor x_4)\]

We put \(k\) to be equal to the number of clauses.
A Reduction from 3-SAT to IS

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4) \land (x_1 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor x_2 \lor x_4)\]

The constructed graph and number \(k\) are the output of the algorithm.
A Reduction from 3-SAT to IS

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4) \land (x_1 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor x_2 \lor x_4)\]

\[v(x_1) = 1\]
\[v(x_2) = 1\]
\[v(x_3) = 0\]
\[v(x_4) = 1\]

If the formula \( \varphi \) is satisfiable then there is a valuation \( v \) where every clause contains at least one literal with value 1.
A Reduction from 3-SAT to IS

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4) \land (x_1 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor x_2 \lor x_4)\]

\[v(x_1) = 1\]
\[v(x_2) = 1\]
\[v(x_3) = 0\]
\[v(x_4) = 1\]

We select one literal that has a value 1 in the valuation \(v\), and we put the corresponding node into the independent set.
A Reduction from 3-SAT to IS

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4) \land (x_1 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor x_2 \lor x_4)\]

\[
v(x_1) = 1
\]
\[
v(x_2) = 1
\]
\[
v(x_3) = 0
\]
\[
v(x_4) = 1
\]

We can easily verify that the selected nodes form an independent set.
The selected nodes form an independent set because:

- One node has been selected from each triple of nodes corresponding to one clause.

- Nodes denoted $x_i$ and $¬x_i$ could not be selected together. (Exactly of them has the value 1 in the given valuation $v$.)

Z. Sawa (TU Ostrava)

Theoretical Computer Science

September 13, 2019

400 / 490
A Reduction from 3-SAT to IS

On the other hand, if there is an independent set of size $k$ in graph $G$, then it surely has the following properties:

- At most one node is selected from each triple of nodes corresponding to one clause.
  But because there are $k$ clauses and $k$ nodes are selected, exactly one node must be selected from each triple.

- Nodes denoted $x_i$ and $\neg x_i$ cannot be selected together.

We can choose a valuation according to the selected nodes, since it follows from the previous discussion that it must exist. (Arbitrary values can be assigned to the remaining variables.)

For the given valuation, the formula $\varphi$ has surely the value 1, since in each clause there is at least one literal with value 1.
A Reduction from 3-SAT to IS

It is obvious that the running time of the described algorithm polynomial: Graph $G$ and number $k$ can be constructed for a formula $\varphi$ in time $O(n^2)$, where $n$ is the size of formula $\varphi$.

We have also seen that there is an independent set of size $k$ in the constructed graph $G$ iff the formula $\varphi$ is satisfiable.

The described algorithm shows that 3-SAT can be reduced in polynomial time to IS.
Complexity Classes

- **PTIME** — the class of all algorithmic problems that can be solved by a (deterministic) algorithm in polynomial time

- **NPTIME** — the class of algorithmic problems that can be solved by a **nondeterministic** algorithm in polynomial time
Let us consider a set of all decision problems.
By an arrow we denote that a problem $A$ can be reduced in polynomial time to a problem $B$. 
For example 3-SAT can be reduced in polynomial time to IS.
Let us consider now the class \textit{NPTIME} and a problem $P$. 

\begin{center}
\includegraphics[width=\textwidth]{np_complete_problems.png}
\end{center}
A problem $P$ is **NP-hard** if every problem from $\text{NPTIME}$ can be reduced in polynomial time to $P$. 
A problem $P$ is **NP-complete** if it is NP-hard and it belongs to the class $\text{NPTIME}$. 
If we have found a polynomial time algorithm for some NP-hard problem $P$, then we would have polynomial time algorithms for all problems $P'$ from NPTIME:

- At first we would apply an algorithm for the reduction from $P'$ to $P$ on an input of a problem $P'$.

- Then we would use a polynomial algorithm for $P$ on the constructed instance of $P$ and returned its result as the answer for the original instance of $P'$.

In such a case, $\text{PTIME} = \text{NPTIME}$ would hold, since for every problem from NPTIME there would be a polynomial-time (deterministic) algorithm.
On the other hand, if there is at least one problem from \textbf{NPTIME} for which a polynomial-time algorithm does not exist, then it means that for none of \textbf{NP}-hard problems there is a polynomial-time algorithm.

It is an open question whether the first or the second possibility holds.
It is not difficult to see that:

If a problem $A$ can be reduced in a polynomial time to a problem $B$ and problem $B$ can be reduced in a polynomial time to a problem $C$, then problem $A$ can be reduced in a polynomial time to problem $C$.

So if we know about some problem $P$ that it is NP-hard and that $P$ can be reduced in a polynomial time to a problem $P'$, then we know that the problem $P'$ is also NP-hard.
Theorem

Problem SAT is NP-complete.

It can be shown that SAT can be reduced in a polynomial time to 3-SAT and we have seen that 3-SAT can be reduced in a polynomial time to IS. This means that problems 3-SAT and IS are NP-hard.

It is not difficult to show that 3-SAT and IS belong to the class NPTIME.

Problems 3-SAT and IS are NP-complete.
NP-Complete Problems

By a polynomial reductions from problems that are already known to be NP-complete, NP-completeness of many other problems can be shown:

- SAT → 3-SAT
- 3-CG
- SUBSET-SUM
- ILP
- 3-SAT → IS
- IS → CLIQUE
- IS → VC
- CLIQUE → HC
- HC → HK
- HK → TSP
Examples of Some NP-Complete Problems

The following previously mentioned problems are NP-complete:

- SAT (boolean satisfiability problem)
- 3-SAT
- IS — independent set problem

On the following slides, examples of some other NP-complete problems are described:

- CG — graph coloring (remark: it is NP-complete even in the special case where we have 3 colors)
- VC — vertex cover
- CLIQUE — clique problem
- HC — Hamiltonian cycle
- HK — Hamiltonian circuit
- TSP — traveling salesman problem
- SUBSET-SUM
- ILP — integer linear programming
Graph Coloring

**Graph coloring**

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

**Question:** Is it possible to color nodes of the graph $G$ using $k$ colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

**Example:** $k = 3$
Graph Coloring

**Graph coloring**

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

**Question:** Is it possible to color nodes of the graph $G$ using $k$ colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

**Example:** $k = 3$

![Graph example](image)

**Answer:** Yes
Graph Coloring

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

**Question:** Is it possible to color nodes of the graph $G$ using $k$ colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

**Example:** $k = 3$
Graph Coloring

Graph coloring

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

**Question:** Is it possible to color nodes of the graph $G$ using $k$ colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

**Example:** $k = 3$

![Graph diagram]

**Answer:** No
**VC – Vertex Cover**

**VC – vertex cover**

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

**Question:** Is there some subset of nodes of $G$ of size $k$ such that every edge has at least one of its nodes in this subset?

**Example:** $k = 6$
VC – Vertex Cover

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

**Question:** Is there some subset of nodes of $G$ of size $k$ such that every edge has at least one of its nodes in this subset?

**Example:** $k = 6$

**Answer:** Yes
CLIQUE

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

**Question:** Is there some subset of nodes of $G$ of size $k$ such that every two nodes from this subset are connected by an edge?

**Example:** $k = 4$
CLIQUE

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

Question: Is there some subset of nodes of $G$ of size $k$ such that every two nodes from this subset are connected by an edge?

Example: $k = 4$

Answer: Yes
Hamiltonian Cycle

HC – Hamiltonian cycle

**Input:** A directed graph $G$.

**Question:** Is there a Hamiltonian cycle in $G$ (i.e., a directed cycle going through each node exactly once)?

**Example:**

![Graph Example](image-url)
Hamiltonian Cycle

HC – Hamiltonian cycle

Input: A directed graph \( G \).

Question: Is there a Hamiltonian cycle in \( G \) (i.e., a directed cycle going through each node exactly once)?

Example:

Answer: No
Hamiltonian Cycle

**HC – Hamiltonian cycle**

**Input:** A directed graph $G$.

**Question:** Is there a Hamiltonian cycle in $G$ (i.e., a directed cycle going through each node exactly once)?

**Example:**

![Diagram of a directed graph](image-url)
Hamiltonian Cycle

**HC – Hamiltonian cycle**

**Input:** A directed graph $G$.

**Question:** Is there a Hamiltonian cycle in $G$ (i.e., a directed cycle going through each node exactly once)?

**Example:**

![Diagram of a directed graph with Hamiltonian cycle]

**Answer:** Yes
Hamiltonian Circuit

HK – Hamiltonian circuit

Input: An undirected graph $G$.

Question: Is there a Hamiltonian circuit in $G$ (i.e., an undirected cycle going through each node exactly once)?

Example:

Answer: No
Input: An undirected graph $G$.

Question: Is there a Hamiltonian circuit in $G$ (i.e., an undirected cycle going through each node exactly once)?

Example:
HK – Hamiltonian circuit

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

**Question:** Is there a Hamiltonian circuit in $G$ (i.e., an undirected cycle going through each node exactly once)?

**Example:**

![Graph Example](image)

**Answer:** Yes
TSP - traveling salesman problem

Input: An undirected graph $G$ with edges labelled with natural numbers and a number $k$.

Question: Is there a closed tour going through all nodes of the graph $G$ such that the sum of labels of edges on this tour is at most $k$?

Example: $k = 70$
Traveling Salesman Problem

**TSP - traveling salesman problem**

**Input:** An undirected graph \( G \) with edges labelled with natural numbers and a number \( k \).

**Question:** Is there a closed tour going through all nodes of the graph \( G \) such that the sum of labels of edges on this tour is at most \( k \)?

**Example:** \( k = 70 \)

![Graph with labeled edges]

**Answer:** \( \text{YES} \), since there is a tour with the sum 69.
Problem SUBSET-SUM

Input: A sequence \(a_1, a_2, \ldots, a_n\) of natural numbers and a natural number \(s\).

Question: Is there a set \(I \subseteq \{1, 2, \ldots, n\}\) such that \(\sum_{i \in I} a_i = s\)?

In other words, the question is whether it is possible to select a subset with sum \(s\) of a given (multi)set of numbers.

**Example:** For the input consisting of numbers 3, 5, 2, 3, 7 and number \(s = 15\) the answer is **Yes**, since \(3 + 5 + 7 = 15\).

For the input consisting of numbers 3, 5, 2, 3, 7 and number \(s = 16\) the answer is **No**, since no subset of these numbers has sum 16.
Remark:
The order of numbers $a_1, a_2, \ldots, a_n$ in an input is not important.

Note that this is not exactly the same as if we have formulated the problem so that the input is a set \{a_1, a_2, \ldots, a_n\} and a number $s$ — numbers cannot occur multiple times in a set but they can in a sequence.
Problem SUBSET-SUM is a special case of a **knapsack problem**:

**Knapsack problem**

**Input:** Sequence of pairs of natural numbers $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ and two natural numbers $s$ and $t$.

**Question:** Is there a set $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in I} a_i \leq s$ and $\sum_{i \in I} b_i \geq t$?
Informally, the knapsack problem can be formulated as follows:

We have $n$ objects, where the $i$-th object weights $a_i$ grams and its price is $b_i$ dollars.

The question is whether there is a subset of these objects with total weight at most $s$ grams ($s$ is the capacity of the knapsack) and with total price at least $t$ dollars.

**Remark:**
Here we have formulated this problem as a decision problem.

This problem is usually formulated as an optimization problem where the aim is to find such a set $I \subseteq \{1, 2, \ldots, n\}$, where the value $\sum_{i \in I} b_i$ is maximal and where the condition $\sum_{i \in I} a_i \leq s$ is satisfied, i.e., where the capacity of the knapsack is not exceeded.
That SUBSET-SUM is a special case of the Knapsack problem can be seen from the following simple construction:

Let us say that $a_1, a_2, \ldots, a_n, s_1$ is an instance of SUBSET-SUM. It is obvious that for the instance of the knapsack problem where we have the sequence $(a_1, a_1), (a_2, a_2), \ldots, (a_n, a_n)$, $s = s_1$ and $t = s_1$, the answer is the same as for the original instance of SUBSET-SUM.
If we want to study the complexity of problems such as SUBSET-SUM or the knapsack problem, we must clarify what we consider as the size of an instance.

Probably the most natural it is to define the size of an instance as the total number of bits needed for its representation. We must specify how natural numbers in the input are represented – if in binary (resp. in some other numeral system with a base at least 2 (e.g., decimal or hexadecimal) or in unary.

- If we consider the total number of bits when numbers are written in **binary** as the size of an input, no polynomial time algorithm is known for SUBSET-SUM.
- If we consider the total number of bits when numbers are written in **unary** as the size of an input, SUBSET-SUM can be solved by an algorithm whose time complexity is polynomial.
Problem ILP (integer linear programming)

Input: An integer matrix $A$ and an integer vector $b$.

Question: Is there an integer vector $x$ such that $Ax \leq b$?

An example of an instance of the problem:

$$
A = \begin{pmatrix}
3 & -2 & 5 \\
1 & 0 & 1 \\
2 & 1 & 0
\end{pmatrix}
\quad b = \begin{pmatrix}
8 \\
-3 \\
5
\end{pmatrix}
$$

So the question is if the following system of inequations has some integer solution:

$$
\begin{align*}
3x_1 - 2x_2 + 5x_3 & \leq 8 \\
x_1 + x_3 & \leq -3 \\
2x_1 + x_2 & \leq 5
\end{align*}
$$
One of solutions of the system

\[
3x_1 - 2x_2 + 5x_3 \leq 8 \\
x_1 + x_3 \leq -3 \\
2x_1 + x_2 \leq 5
\]

is for example \( x_1 = -4, x_2 = 1, x_3 = 1 \), i.e.,

\[
x = \begin{pmatrix} -4 \\ 1 \\ 1 \end{pmatrix}
\]

because

\[
3 \cdot (-4) - 2 \cdot 1 + 5 \cdot 1 = -9 \leq 8 \\
-4 + 1 = -3 \leq -3 \\
2 \cdot (-4) + 1 = -7 \leq 5
\]

So the answer for this instance is \textbf{YES}.
Remark: A similar problem where the question for a given system of linear inequations is whether it has a solution in the set of real numbers, can be solved in a polynomial time.
Undecidable Problems
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 $P$).
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.
Let us consider some general programming language $L$.

Furthermore, let us assume that programs in language $L$ run on some idealized machine where a (potentially) unbounded amount of memory is available — i.e., the allocation of memory never fails.

**Example:** The following problem called the **Halting problem** is undecidable:

**Halting problem**

**Input:** A source code of a $L$ program $P$, input data $x$.

**Question:** Does the computation of $P$ on the input $x$ halt after some finite number of steps?
Let us assume that there is a program that can decide the Halting problem. So we could construct a subroutine $H$, declared as

$$\text{Bool } H(\text{String code, String input})$$

where $H(P, x)$ returns:
- true if the program $P$ halts on the input $x$,
- false if the program $P$ does not halt on the input $x$.

Remark: Let us say that subroutine $H(P, x)$ returns false if $P$ is not a syntactically correct program.
Halting Problem

Using the subroutine $H$ we can construct a program $D$ that performs the following steps:

- It reads its input into a variable $x$ of type $\text{String}$.
- It calls the subroutine $H(x, x)$.
- If subroutine $H$ returns $\text{true}$, program $D$ jumps into an infinite loop

```
loop: goto loop
```

In case that $H$ returns $\text{false}$, program $D$ halts.

What does the program $D$ do if it gets its own code as an input?
Halting Problem

If $D$ gets its own code as an input, it either halts or not.

- If $D$ halts then $H(D, D)$ returns true and $D$ jumps into the infinite loop. A contradiction!
- If $D$ does not halt then $H(D, D)$ returns false and $D$ halts. A contradiction!

In both case we obtain a contradiction and there is no other possibility. So the assumption that $H$ solves the Halting problem must be wrong.
Reduction between Problems

If we have already proved a (decision) problem to be undecidable, we can prove undecidability of other problems by reductions.

Problem $P_1$ can be reduced to problem $P_2$ if there is an algorithm $Alg$ such that:

- It can get an arbitrary instance of problem $P_1$ as an input.
- For an instance of a problem $P_1$ obtained as an input (let us denote it as $w$) it produces an instance of a problem $P_2$ as an output.
- It holds i.e., the answer for the input $w$ of problem $P_1$ is $\text{YES}$ iff the answer for the input $Alg(w)$ of problem $P_2$ is $\text{YES}$. 
Reductions between Problems

Inputs of problem $P_1$

Inputs of problem $P_2$
Reductions between Problems

Inputs of problem $P_1$ to Inputs of problem $P_2$

Alg
Let us say there is some reduction $Alg$ from problem $P_1$ to problem $P_2$.

If problem $P_2$ is decidable then problem $P_1$ is also decidable.

Solution of problem $P_1$ for an input $x$:

- Call $Alg$ with $x$ as an input, it returns a value $Alg(x)$.
- Call the algorithm solving problem $P_2$ with input $Alg(x)$.
- Write the returned value to the output as the result.

It is obvious that if $P_1$ is undecidable then $P_2$ cannot be decidable.
By reductions from the Halting problem we can show undecidability of many other problems dealing with a behaviour of programs:

- Is for some input the output of a given program Yes?
- Does a given program halt for an arbitrary input?
- Do two given programs produce the same outputs for the same inputs?
- ...
For the use in proofs and in reductions between problems, it is convenient to have the language $\mathcal{L}$ and the machine running programs in this language as simple as possible:

- the number of kinds of instructions as small as possible
- instructions as primitive as possible
- the datatypes, with which the algorithm works, as simple as possible
- it is irrelevant how difficult is to write programs in the given language (it can be extremely user-unfriently)

On the other hand, such language (resp. machine) must be general enough so that any program written in an arbitrary programming language can be compiled to it.
Models of Computation

Such languages (resp. machines), which are general enough, so that programs written in any other programming language can be translated to them, are called **Turing complete**.

Examples of such Turing complete **models of computation** (languages or machines) often used in proofs:

- Turing machine (Alan Turing)
- Lambda calculus (Alonzo Church)
- Minsky machine (Marvin Minsky)
- ...
Turing machine:

- Let us extend a deterministic finite automaton in the following way:
  - the reading head can move in both directions
  - it is possible to write symbols on the tape
  - the tape is extended into infinity
Church-Turing thesis

Every algorithm can be implemented as a Turing machine.

It is not a theorem that can be proved in a mathematical sense – it is not formally defined what an algorithm is.

The thesis was formulated in 1930s independently by Alan Turing and Alonzo Church.
For purposes of proofs, the following version of Halting problem is often used:

**Halting problem**

**Input:** A description of a Turing machine $M$ and a word $w$.

**Question:** Does the computation of the machine $M$ on the word $w$ halt after some finite number of steps?
Other Undecidable Problems

We have already seen the following example of an undecidable problem:

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Context-free grammars $G_1$ and $G_2$.</td>
</tr>
<tr>
<td><strong>Question:</strong> Is $L(G_1) = L(G_2)$?</td>
</tr>
</tbody>
</table>

respectively

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A context-free grammar generating a language over an alphabet $\Sigma$.</td>
</tr>
<tr>
<td><strong>Question:</strong> Is $L(G) = \Sigma^*$?</td>
</tr>
</tbody>
</table>
An input is a set of types of tiles, such as:

![Tiles](image)

The question is whether it is possible to cover every finite area of an arbitrary size using the given types of tiles in such a way that the colors of neighboring tiles agree.

**Remark:** We can assume that we have an infinite number of tiles of all types.

The tiles cannot be rotated.
Other Undecidable Problems
Other Undecidable Problems

An input is a set of types of cards, such as:

```
abb
bbab

a
aa

bab
ab

baba
aa

aba
a
```

The question is whether it is possible to construct from the given types of cards a non-empty finite sequence such that the concatenations of the words in the upper row and in the lower row are the same. Every type of a card can be used repeatedly.

```
a
aa

abb
bbab

abb
bbab

baba
aa

abb
bbab

aba
a
```

In the upper and in the lower row we obtained the word

```
aabbabbbabaabbaba.
```
Undecidability of several other problems dealing with context-free grammars can be proved by reductions from the previous problem:

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Context-free grammars $G_1$ and $G_2$.</td>
</tr>
<tr>
<td><strong>Question:</strong> Is $L(G_1) \cap L(G_2) = \emptyset$?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A context-free grammar $G$.</td>
</tr>
<tr>
<td><strong>Question:</strong> Is $G$ ambiguous?</td>
</tr>
</tbody>
</table>
### Problem

**Input:** A closed formula of the first order predicate logic where the only predicate symbols are $=$ and $<$, the only function symbols are $+$ and $\ast$, and the only constant symbols are $0$ and $1$.

**Question:** Is the given formula true in the domain of natural numbers (using the natural interpretation of all function and predicate symbols)?

An example of an input:

$$\forall x \exists y \forall z ((x \ast y = z) \land (y + 1 = x))$$

**Remark:** There is a close connection with Gödel's incompleteness theorem.
It is interesting that an analogous problem, where real numbers are considered instead of natural numbers, is decidable (but the algorithm for it and the proof of its correctness are quite nontrivial).

Also when we consider natural numbers or integers and the same formulas as in the previous case but with the restriction that it is not allowed to use the multiplication function symbol $\ast$, the problem is algorithmically decidable.
Other Undecidable Problems

If the function symbol $\ast$ can be used then even the very restricted case is undecidable:

**Hilbert’s tenth problem**

**Input:** A polynomial $f(x_1, x_2, \ldots, x_n)$ constructed from variables $x_1, x_2, \ldots, x_n$ and integer constants.

**Question:** Are there some natural numbers $x_1, x_2, \ldots, x_n$ such that $f(x_1, x_2, \ldots, x_n) = 0$?

An example of an input: $5x^2y - 8yz + 3z^2 - 15$

I.e., the question is whether

$$\exists x \exists y \exists z (5 \ast x \ast x \ast y + (-8) \ast y \ast z + 3 \ast z \ast z + (-15) = 0)$$

holds in the domain of natural numbers.
Also the following problem is algorithmically undecidable:

**Problem**

- **Input:** A closed formula $\varphi$ of the first-order predicate logic.
- **Question:** Is $\models \varphi$?

**Remark:** Notation $\models \varphi$ denotes that formula $\varphi$ is logically valid, i.e., it is true in all interpretations.
So far we have considered only the time necessary for a computation. Sometimes the size of the memory necessary for the computation is more critical.

The **amount of memory** used by machine $\mathcal{M}$ in a computation on input $w$ can be for example:

- the maximal number of bits necessary for storing all data for each configuration
- the maximal number of memory cells used during the computation

**Definition**

A **space complexity** of algorithm $Alg$ running on machine $\mathcal{M}$ is the function $S : \mathbb{N} \rightarrow \mathbb{N}$, where $S(n)$ is the maximal amount of memory used by $\mathcal{M}$ for inputs of size $n$. 
Space Complexity of Algorithms

- There can be two algorithms for a particular problem such that one of them has a smaller time complexity and the other a smaller space complexity.

- If the time-complexity of an algorithm is in $O(f(n))$ then also the space complexity is in $O(f(n))$ (note that the number of memory cells used in one instruction is bounded by some constant that does not depend on the size of an input).

- The space complexity can be much smaller than the time complexity — the space complexity of INSERTION-SORT is $\Theta(n)$, while its time complexity is $\Theta(n^2)$.
Some typical values of the size of an input $n$, for which an algorithm with the given time complexity usually computes the output on a “common PC” within a fraction of a second or at most in seconds. (Of course, this depends on particular details. Moreover, it is assumed here that no big constants are hidden in the asymptotic notation.)

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Typical Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n)$</td>
<td>$1,000,000 - 1,000,000,000$</td>
</tr>
<tr>
<td>$O(n \log n)$</td>
<td>$100,000 - 1,000,000$</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>$1000 - 10,000$</td>
</tr>
<tr>
<td>$O(n^3)$</td>
<td>$100 - 1000$</td>
</tr>
<tr>
<td>$2^{O(n)}$</td>
<td>$20 - 30$</td>
</tr>
<tr>
<td>$O(n!)$</td>
<td>$10 - 15$</td>
</tr>
</tbody>
</table>
When we use asymptotic estimations of the complexity of algorithms, we should be aware of some issues:

- Asymptotic estimations describe only how the running time grows with the growing size of input instance.
- They do not say anything about exact running time. Some big constants can be hidden in the asymptotic notation.
- An algorithm with better asymptotic complexity than some other algorithm can be in reality faster only for very big inputs.
- We usually analyze the time complexity in the worst case. For some algorithms, the running time in the worst case can be much higher than the running time on “typical” instances.
This can be illustrated on algorithms for sorting.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Worst-case</th>
<th>Average-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubblesort</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>Heapsort</td>
<td>$\Theta(n \log n)$</td>
<td>$\Theta(n \log n)$</td>
</tr>
<tr>
<td>Quicksort</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n \log n)$</td>
</tr>
</tbody>
</table>

Quicksort has a worse asymptotic complexity in the worst case than Heapsort and the same asymptotic complexity in an average case but it is usually faster in practice.
**Polynomial** — an expression of the form

\[ a_k n^k + a_{k-1} n^{k-1} + \cdots + a_2 n^2 + a_1 n + a_0 \]

where \( a_0, a_1, \ldots, a_k \) are constants.

Examples of polynomials:

\[ 4n^3 - 2n^2 + 8n + 13 \quad 2n + 1 \quad n^{100} \]

Function \( f \) is called **polynomial** if it is bounded from above by some polynomial, i.e., if there exists a constant \( k \) such that \( f \in O(n^k) \).

For example, the functions belonging to the following classes are polynomial:

\[ O(n) \quad O(n \log n) \quad O(n^2) \quad O(n^5) \quad O(\sqrt{n}) \quad O(n^{100}) \]
Complexity of Algorithms

Function such as $2^n$ or $n!$ are not polynomial — for arbitrarily big constant $k$ we have

$$2^n \in \Omega(n^k) \quad n! \in \Omega(n^k)$$

**Polynomial algorithm** — an algorithm whose time complexity is polynomial (i.e., bounded from above by some polynomial)

Roughly we can say that:

- polynomial algorithms are efficient algorithms that can be used in practice for inputs of considerable size
- algorithms, which are not polynomial, can be used in practice only for rather small inputs
The division of algorithms on polynomial and non-polynomial is very rough — we cannot claim that polynomial algorithms are always efficient and non-polynomial algorithms are not:

- an algorithm with the time complexity $\Theta(n^{100})$ is probably not very useful in practice,

- some algorithms, which are non-polynomial, can still work very efficiently for majority of inputs, and can have a time complexity bigger than polynomial only due to some problematic inputs, on which the computation takes long time.

**Remark:** Polynomial algorithms where the constant in the exponent is some big number (e.g., algorithms with complexity $\Theta(n^{100})$) almost never occur in practice as solutions of usual algorithmic problems.
Complexity of Algorithms

For most of common algorithmic problems, one of the following three possibilities happens:

- A polynomial algorithm with time complexity $O(n^k)$ is known, where $k$ is some very small number (e.g., 5 or more often 3 or less).

- No polynomial algorithm is known and the best known algorithms have complexities such as $2^{\Theta(n)}$, $\Theta(n!)$, or some even bigger.

In some cases, a proof is known that there does not exist a polynomial algorithm for the given problem (it cannot be constructed).

- No algorithm solving the given problem is known (and it is possibly proved that there does not exist such algorithm).
A typical example of polynomial algorithm — matrix multiplication with time complexity $\Theta(n^3)$ and space complexity $\Theta(n^2)$:

Algorithm 3: Matrix multiplication

1. `MATRIX-MULT (A, B, C, n):
2. begin
3.   for $i := 1$ to $n$ do
4.     for $j := 1$ to $n$ do
5.       $x := 0$
6.       for $k := 1$ to $n$ do
7.         $x := x + A[i][k] * B[k][j]$
8.       end
9.     $C[i][j] := x$
10. end
11. end
12. end
Complexity of Algorithms

- For a rough estimation of complexity, it is often sufficient to count the number of nested loops — this number then gives the degree of the polynomial.

**Example:** Three nested loops in the matrix multiplication — the time complexity of the algorithm is $O(n^3)$.

- If it is not the case that all the loops go from 0 to $n$ but the number of iterations of inner loops are different for different iterations of an outer loops, a more precise analysis can be more complicated. It is often the case, that the sum of some sequence (e.g., the sum of arithmetic or geometric progression) is then computed in the analysis.

The results of such more detailed analysis often does not differ from the results of a rough analysis but in many cases the time complexity resulting from a more detailed analysis can be considerably smaller than the time complexity following from the rough analysis.
**Arithmetic progression** — a sequence of numbers $a_0, a_1, \ldots, a_{n-1}$, where

$$a_i = a_0 + i \cdot d,$$

where $d$ is some constant independent on $i$.

**Remark:** So in an arithmetic progression, we have $a_{i+1} = a_i + d$ for each $i$.

**The sum of an arithmetic progression:**

$$\sum_{i=0}^{n-1} a_i = a_0 + a_1 + \cdots + a_{n-1} = \frac{1}{2}n(a_{n-1} + a_0)$$
Example:

\[ 1 + 2 + \cdots + n = \frac{1}{2} n(n + 1) = \frac{1}{2} n^2 + \frac{1}{2} n = \Theta(n^2) \]

For example, for \( n = 100 \) we have

\[ 1 + 2 + \cdots + 100 = 50 \cdot 101 = 5050. \]

Remark: To see this, we can note that

\[ 1 + 2 + \cdots + 100 = (1 + 100) + (2 + 99) + \cdots + (50 + 51), \]

where we compute the sum of 50 pairs of number, where the sum of each pair is 101.
**Geometric progression** — a sequence of numbers \( a_0, a_1, \ldots, a_n \), where

\[
a_i = a_0 \cdot q^i,
\]

where \( q \) is some constant independent on \( i \).

**Remark:** So in a geometric progression we have \( a_{i+1} = a_i \cdot q \).

**The sum of a geometric progression** (where \( q \neq 1 \)):

\[
\sum_{i=0}^{n} a_i = a_0 + a_1 + \cdots + a_n = a_0 \frac{q^{n+1} - 1}{q - 1}
\]
Example:

\[ 1 + q + q^2 + \cdots + q^n = \frac{q^{n+1} - 1}{q - 1} \]

In particular, for \( q = 2 \):

\[ 1 + 2^1 + 2^2 + 2^3 + \cdots + 2^n = \frac{2^{n+1} - 1}{2 - 1} = 2 \cdot 2^n - 1 = \Theta(2^n) \]
An **exponential** function: a function of the form $c^n$, where $c$ is a constant — e.g., function $2^n$

**Logarithm** — the inverse function to an exponential function: for a given $n$,

$$\log_c n$$

is the value $x$ such that $c^x = n$. 

<table>
<thead>
<tr>
<th>$n$</th>
<th>$2^n$</th>
<th>$n$</th>
<th>$\lceil \log_2 n \rceil$</th>
<th>$n$</th>
<th>$\log_2 n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>—</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>4</td>
<td>2</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>5</td>
<td>3</td>
<td>32</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>6</td>
<td>3</td>
<td>64</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>7</td>
<td>3</td>
<td>128</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>8</td>
<td>3</td>
<td>256</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
<td>9</td>
<td>4</td>
<td>512</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
<td>10</td>
<td>4</td>
<td>1024</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>2048</td>
<td>11</td>
<td>4</td>
<td>2048</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>4096</td>
<td>12</td>
<td>4</td>
<td>4096</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>8192</td>
<td>13</td>
<td>4</td>
<td>8192</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>16384</td>
<td>14</td>
<td>4</td>
<td>16384</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>32768</td>
<td>15</td>
<td>4</td>
<td>32768</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>65536</td>
<td>16</td>
<td>4</td>
<td>65536</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>131072</td>
<td>17</td>
<td>5</td>
<td>131072</td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>262144</td>
<td>18</td>
<td>5</td>
<td>262144</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>524288</td>
<td>19</td>
<td>5</td>
<td>524288</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>1048576</td>
<td>20</td>
<td>5</td>
<td>1048576</td>
<td>20</td>
</tr>
</tbody>
</table>
Proposition

For any $a, b > 1$ and any $n > 0$ we have

$$\log_a n = \frac{\log_b n}{\log_b a}$$

Proof: From $n = a^{\log_a n}$ it follows that $\log_b n = \log_b (a^{\log_a n})$.

Since $\log_b (a^{\log_a n}) = \log_a n \cdot \log_b a$, we obtain $\log_b n = \log_a n \cdot \log_b a$, from which the above mentioned conclusion follows directly.

Due to this observation, the base of a logarithm is often omitted in the asymptotic notation: for example, instead of $\Theta(n \log_2 n)$ we can write $\Theta(n \log n)$. 
Examples where exponential functions and logarithms can appear in an analysis of algorithms:

- Some value is repeatedly decreased to one half or is repeatedly doubled.

For example, in the binary search, the size of an interval halves in every iteration of the loop.

Let us assume that an array has size $n$.

What is the minimal size of an array $n$, for which the algorithm performs at least $k$ iterations?

The answer: $2^k$

So we have $k = \log_2(n)$. The time complexity of the algorithm is then $\Theta(\log n)$. 
Using $n$ bits we can represent numbers from 0 to $2^n - 1$.

The minimal numbers of bits, which are sufficient for representing a natural number $x$ in binary is

$$\lceil \log_2 (x + 1) \rceil.$$  

A perfectly balanced tree of height $h$ has $2^{h+1} - 1$ nodes, and $2^h$ of these nodes are leaves.

The height of a perfectly balanced binary tree with $n$ nodes is $\log_2 n$.

An illustrating example: If we would draw a balanced tree with $n = 1\,000\,000$ nodes in such a way that the distance between neighbouring nodes would be 1 cm and the height of each layer of nodes would be also 1 cm, the width of the tree would be 10 km and its height would be approximately 20 cm.
A perfectly balanced binary tree of height $h$:
A perfectly balanced binary tree of height $h$:
Example: Algorithm **Merge-Sort**.

The main idea of the algorithm: Two sorted sequences can be easily merged into one sorted sequence. If both sequences have together *n* elements then this operation can be done in *n* steps.
**Example:** Algorithm **Merge-Sort**.

The main idea of the algorithm: Two sorted sequences can be easily merged into one sorted sequence. If both sequences have together $n$ elements then this operation can be done in $n$ steps.

\[
\begin{array}{cccc}
34 & 42 & 58 & 61 \\
11 & 53 & 67 \\
\end{array}
\Rightarrow
\begin{array}{c}
10 \\
\end{array}
\]
Complexity of Algorithms

**Example:** Algorithm **Merge-Sort**.

The main idea of the algorithm: Two sorted sequences can be easily merged into one sorted sequence.

If both sequences have together $n$ elements then this operation can be done in $n$ steps.

\[
\begin{array}{c}
34 & 42 & 58 & 61 \\
53 & 67 \\
\end{array} \quad \Rightarrow \quad \begin{array}{c}
10 & 11
\end{array}
\]
Example: Algorithm **Merge-Sort**.

The main idea of the algorithm: Two sorted sequences can be easily merged into one sorted sequence. If both sequences have together $n$ elements then this operation can be done in $n$ steps.
Example: Algorithm **Merge-Sort**.

The main idea of the algorithm: Two sorted sequences can be easily merged into one sorted sequence. If both sequences have together $n$ elements then this operation can be done in $n$ steps.

\[
\begin{array}{c}
58 & 61 \\
53 & 67 \\
\end{array}
\quad \Rightarrow 
\begin{array}{c}
10 & 11 & 34 & 42 \\
\end{array}
\]
**Example:** Algorithm **Merge-Sort**.

The main idea of the algorithm: Two sorted sequences can be easily merged into one sorted sequence.

If both sequences have together $n$ elements then this operation can be done in $n$ steps.

\[
\begin{align*}
58 & \quad 61 \\
67 & \quad \Rightarrow \\
10 & \quad 11 & \quad 34 & \quad 42 & \quad 53
\end{align*}
\]
Example: Algorithm **Merge-Sort**.

The main idea of the algorithm: Two sorted sequences can be easily merged into one sorted sequence. If both sequences have together \( n \) elements then this operation can be done in \( n \) steps.
Example: Algorithm \textsc{Merge-Sort}.

The main idea of the algorithm: Two sorted sequences can be easily merged into one sorted sequence.
If both sequences have together $n$ elements then this operation can be done in $n$ steps.
Example: Algorithm **Merge-Sort**.

The main idea of the algorithm: Two sorted sequences can be easily merged into one sorted sequence. If both sequences have together \( n \) elements then this operation can be done in \( n \) steps.

\[\begin{array}{cccccccc}
10 & 11 & 34 & 42 & 53 & 58 & 61 & 67
\end{array}\]
**Complexity of Algorithms**

---

**Algorithm 4: Merge sort**

1. **Merge-Sort** \((A, p, r):\)
   2. begin
   3. \(\text{if } r - p > 1 \text{ then}\)
   4. \(q := \lfloor (p + r) / 2 \rfloor\)
   5. **Merge-Sort**\((A, p, q)\)
   6. **Merge-Sort**\((A, q, r)\)
   7. **Merge**\((A, p, q, r)\)
   8. end
   9. end

To sort an array \(A\) containing elements \(A[0], A[1], \cdots, A[n - 1]\) we call **Merge-Sort**\((A, 0, n)\).

**Remark:** Procedure **Merge**\((A, p, q, r)\) merges sorted sequences stored in \(A[p \ldots q - 1]\) and \(A[q \ldots r - 1]\) into one sequence stored in \(A[p \ldots r - 1]\).
The tree of recursive calls has $\Theta(\log n)$ layers. On each layer, $\Theta(n)$ operations are performed. The time complexity of \textsc{Merge-Sort} is $\Theta(n \log n)$. 

Input: 58, 42, 34, 61, 67, 10, 53, 11
Complexity of Algorithms

Representations of a graph:

1 2 3
4 5 6

1 2 3 4 5
6

Z. Sawa (TU Ostrava)
Theoretical Computer Science
September 13, 2019 475 / 490
Representations of a graph:
Finding the shortest path in a graph where edges are not weighted:

- Breadth-first search
- The input is a graph $G$ (with a set of nodes $V$) and an initial node $s$.
- The algorithm finds the shortest paths from node $s$ for all nodes.
- For a graph with $n$ nodes and $m$ edges, the running time of the algorithm is $\Theta(n + m)$. 

Z. Sawa (TU Ostrava) Theoretical Computer Science September 13, 2019 477 / 490
Algorithm 5: Breadth-first search

1 \textbf{Bfs} \((G, s)\):
2 \hspace{1em} \textbf{begin}
3 \hspace{2em} \textbf{Bfs-Init} \((G, s)\)
4 \hspace{2em} \textbf{Enqueue} \((Q, s)\)
5 \hspace{2em} \textbf{while} \(Q \neq \emptyset\) \textbf{do}
6 \hspace{3em} \textbf{u} := \textbf{Dequeue} \((Q)\)
7 \hspace{3em} \textbf{for each} \(v \in \text{edges}[u]\) \textbf{do}
8 \hspace{4em} \textbf{if} \(\text{color}[v] = \text{WHITE}\) \textbf{then}
9 \hspace{5em} \text{color}[v] := \text{GRAY}
10 \hspace{5em} \text{d}[v] := \text{d}[u] + 1
11 \hspace{5em} \text{pred}[v] := u
12 \hspace{5em} \textbf{Enqueue} \((Q, v)\)
13 \hspace{4em} \textbf{end}
14 \hspace{3em} \textbf{end}
15 \hspace{2em} \text{color}[u] := \text{BLACK}
16 \hspace{2em} \textbf{end}
17 \textbf{end}
Algorithm 6: Breadth-first search — initialization

1 \textbf{Bfs-Init} \((G, s)\):
2 \textbf{begin}
3 \quad \textbf{for} each \(u \in V - \{s\} \textbf{ do}
4 \quad \quad \text{color}[u] := \text{WHITE}
5 \quad \quad d[u] := \infty
6 \quad \quad pred[u] := \text{NIL}
7 \quad \textbf{end}
8 \quad color[s] := \text{GRAY}
9 \quad d[s] := 0
10 \quad pred[s] := \text{NIL}
11 \quad Q := \emptyset
12 \textbf{end}
Other Examples of Problems

Problem “Primality”

- **Input:** A natural number $x$.
- **Output:** Yes if $x$ is a prime, No otherwise.

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

Few of the first primes: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, . . .
The problems, where the set of outputs is \{Yes, 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 \(x\).

**Question:** Is \(x\) a prime?
A simple algorithm solving the “Primality” problem can work like this:

- Test the trivial cases (e.g., if $x \leq 2$ or if $x$ is even).
- Try to divide $x$ successively by all odd numbers in interval $3, \ldots, \lfloor \sqrt{x} \rfloor$.

Let $n$ be the number of bits in the representation of number $x$, e.g., $n = \lceil \log_2(x + 1) \rceil$. This value $n$ will be considered as the size of the input.

Note that the number $\lfloor \sqrt{x} \rfloor$ has approximately $n/2$ bits.

There are approximately $2^{(n/2) - 1}$ odd numbers in the interval $3, \ldots, \lfloor \sqrt{x} \rfloor$, and so the time complexity of this simple algorithm is in $2^{\Theta(n)}$. 
Primality Test

This simple algorithm with an exponential running time (resp. also different improved versions of this) are applicable to numbers with thousands of bits in practice. A primality test of such big numbers plays an important role for example in cryptography.

Only since 2003, a polynomial time algorithm is known. The time complexity of the original version of the algorithm was $O(n^{12+\varepsilon})$, later it was improved to $(O(n^{7.5}))$. The currently fastest algorithm has time complexity $O(n^6)$.

In practice, randomized algorithms are used for primality testing:

- Solovay–Strassen
- Miller–Rabin

(The time complexity of both algorithms is $O(n^3)$.)
A **randomized algorithm**:  
- It uses a random-number generator during a computation.  
- It can produce different outputs in different runs with the same input.  
- The output need not be always correct but the probability of producing an incorrect output is bounded.

For example, both above mentioned randomized algorithms for primality testing behave as follows:

- If \( x \) is a prime, the answer **Yes** is always returned.  
- If \( x \) is not a prime, the probability of the answer **No** is at least 50% but there is at most 50% probability that the program returns the incorrect answer **Yes**.
The program can be run repeatedly ($k$ times):

- If the program returns at least once the answer **No**, we know (with 100% probability) that $x$ is not a prime.
- If the program always returns **Yes**, the probability that $x$ is not a prime is at most $\frac{1}{2^k}$.

For sufficiently large values of $k$, the probability of an incorrect answer is negligible.

**Remark:** For example for $k = 100$, the probability of this error is smaller than the probability that a computer, on which the program is running, will be destroyed by a falling meteorite (assuming that at least once in every 1000 years at least $100\, \text{m}^2$ of Earth surface is destroyed by a meteorite).
At first sight, the following problem looks very similar as the primality test:

**Problem “Factorization”**

**Input:** A natural number $x$, where $x > 1$.
**Output:** Primes $p_1, p_2, \ldots, p_m$ such that $x = p_1 \cdot p_2 \cdot \cdots \cdot p_m$.

In fact, this problem is (supposed to be) much harder than primality testing.
No efficient (polynomial) algorithm is known for this problem (nor a randomized algorithm).
### Independent set (IS) problem

<table>
<thead>
<tr>
<th>Input:</th>
<th>An undirected graph $G$, a number $k$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question:</td>
<td>Is there an independent set of size $k$ in the graph $G$?</td>
</tr>
</tbody>
</table>

**Remark:** An **independent set** in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.
Other Examples of Problems

Independent set (IS) problem

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

**Question:** Is there an independent set of size $k$ in the graph $G$?

Remark: An **independent set** in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.
Other Examples of Problems

An example of an instance where the answer is **YES**:

![Graph with $k = 4$]

An example of an instance where the answer is **NO**:

![Graph with $k = 5$]
Other Examples of Problems

- A set containing $n$ elements has $2^n$ subsets.
  
  Consider for example an algorithm solving a given problem by brute force where it tests the required property for each subset of a given set.
  
- It is sufficient to consider only subsets of size $k$. The total number of such subsets is
  
  $$\binom{n}{k}$$

  For some values of $k$, the total number of these subsets is not much smaller than $2^n$:

  For example, it is not too difficult to show that

  $$\binom{n}{\lfloor n/2 \rfloor} \geq \frac{2^n}{n}.$$
Let us have an algorithm solving the independent set problem by brute force in such a way that it tests for each subset with $k$ elements of the set of nodes (with $n$ nodes), if it forms an independent set. The time complexity of the algorithm is $2^{\Theta(n)}$. 