Introduction to Theoretical Computer Science

Zdeněk Sawa

Department of Computer Science, FEI, Technical University of Ostrava
17. listopadu 15, Ostrava-Poruba 708 33
Czech republic

February 6, 2014
Name: Ing. Zdeněk Sawa, Ph.D.
E-mail: zdenek.sawa@vsb.cz
Room: A1024

On these pages you will find:

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

- **Credit** (22 points):
  - Written test (22 points) — it will be written on a tutorial

  The minimal requirement for obtaining the credit is 7 points.

  A correcting test for 14 points.

- **Exam** (78 points)
  - A written exam consisting of three parts (26 points for each part); it is necessary to obtain at least 10 points for each part.
Different areas of theoretical computer science:

- algorithms
- computational complexity
- models of computation
- automata theory
- formal languages
- syntax and semantics of programming languages
- type theory
- parallel and distributed systems
- ...
Overlapping with many other areas of mathematics and computer science:

- logic
- graph theory
- number theory
- cryptography
- computational geometry
- game theory
- numerical mathematics
- ...
Some important characteristics of theoretical computer science:

- formal mathematical approach to problems
- the use of mathematical definitions and proofs
- rigorous mathematical proofs

The notion of a proof, axiomatic approach.

Logic — the study of reasoning, proofs, formalisms
Conjunction: \( A \land B \)

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>( A \land B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Remark: also denoted as \& or and
Rules for Conjunction

\(\land i:\) \quad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \land B}

\(\land e_1:\) \quad \frac{\Gamma \vdash A \land B}{\Gamma \vdash A}

\(\land e_2:\) \quad \frac{\Gamma \vdash A \land B}{\Gamma \vdash B}

\text{Assm:} \quad \frac{}{\Gamma, A, \Delta \vdash A}
A Formally Written Proof

\[ A \land B, C \land D \vdash B \land C \]

1. \( A \land B, C \land D \vdash A \land B \)  (Assm)
2. \( A \land B, C \land D \vdash B \)  (\( \land e_2 \) 1)
3. \( A \land B, C \land D \vdash C \land D \)  (Assm)
4. \( A \land B, C \land D \vdash C \)  (\( \land e_1 \) 3)
5. \( A \land B, C \land D \vdash B \land C \)  (\( \land i \) 2,4)
A Formally Written Proof

\[ A \land B, \ C \land D \vdash B \land C \]

1. \( \Gamma \vdash A \land B \) (Assm)
2. \( \Gamma \vdash C \land D \) (Assm)
3. \( \Gamma \vdash C \) (\( \land e_1 \ 2 \))
4. \( \Gamma \vdash B \) (\( \land e_2 \ 1 \))
5. \( \Gamma \vdash B \land C \) (\( \land i \ 4,3 \))

Remark: \( \Gamma := A \land B, \ C \land D \)
A Proof Represented as a Tree

\[ A \land B, C \land D \vdash B \land C \]

\[ \frac{\Gamma \vdash A \land B \quad \Gamma \vdash C \land D}{\Gamma \vdash B \quad \Gamma \vdash C \quad \Gamma \vdash B \land C} \]

Remark: \[ \Gamma := A \land B, C \land D \]
Derived Rule

\[ \Gamma \vdash (A \land B) \land C \]

\[ \Gamma \vdash A \land (B \land C) \]

1. \( \Gamma \vdash (A \land B) \land C \) (premise)
2. \( \Gamma \vdash A \land B \) \( (\land e_1 \ 1) \)
3. \( \Gamma \vdash A \) \( (\land e_1 \ 2) \)
4. \( \Gamma \vdash B \) \( (\land e_2 \ 2) \)
5. \( \Gamma \vdash C \) \( (\land e_2 \ 1) \)
6. \( \Gamma \vdash B \land C \) \( (\land i \ 4,5) \)
7. \( \Gamma \vdash A \land (B \land C) \) \( (\land i \ 3,6) \)
Assumptions

permutation: 
\[ \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C} \]

weakening: 
\[ \frac{\Gamma, \Delta \vdash B}{\Gamma, A, \Delta \vdash B} \]

contraction: 
\[ \frac{\Gamma, A, A, \Delta \vdash B}{\Gamma, A, \Delta \vdash B} \]

Ant: 
\[ \frac{\Gamma \vdash A}{\Gamma' \vdash A} (\Gamma \subseteq \Gamma') \]

Ch: 
\[ \frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B} \]
Disjunction: $A \lor B$

\[
\begin{array}{ccc}
A & B & A \lor B \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

**Remark:** also denoted as **or**
**XOR**

**xor (exclusive or):** $A \oplus B$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$A \oplus B$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Rules for Disjunction

\[ \forall i_1: \frac{\Gamma \vdash A}{\Gamma \vdash A \lor B} \quad \forall i_2: \frac{\Gamma \vdash B}{\Gamma \vdash A \lor B} \]

\[ \forall e: \frac{\Gamma \vdash A \lor B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \]
An Example of a Proof

\[(A \lor B) \lor C \vdash A \lor (B \lor C)\]

1. \(\varphi \vdash (A \lor B) \lor C\) \hspace{1cm} (Assm)
2. \(\varphi, A \lor B \vdash A \lor B\) \hspace{1cm} (Assm)
3. \(\varphi, A \lor B, A \vdash A\) \hspace{1cm} (Assm)
4. \(\varphi, A \lor B, A \vdash A \lor (B \lor C)\) \hspace{1cm} (\lori 3)
5. \(\varphi, A \lor B, B \vdash B\) \hspace{1cm} (Assm)
6. \(\varphi, A \lor B, B \vdash B \lor C\) \hspace{1cm} (\lori 5)
7. \(\varphi, A \lor B, B \vdash A \lor (B \lor C)\) \hspace{1cm} (\lori 6)
8. \(\varphi, A \lor B \vdash A \lor (B \lor C)\) \hspace{1cm} (\lori 2,4,7)
9. \(\varphi, C \vdash C\) \hspace{1cm} (Assm)
10. \(\varphi, C \vdash B \lor C\) \hspace{1cm} (\lori 9)
11. \(\varphi, C \vdash A \lor (B \lor C)\) \hspace{1cm} (\lori 10)
12. \(\varphi \vdash A \lor (B \lor C)\) \hspace{1cm} (\lori 1,8,11)

Remark: \(\varphi := (A \lor B) \lor C\)
Negation: $\neg A$

<table>
<thead>
<tr>
<th></th>
<th>$A$</th>
<th>$\neg A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Remark: also denoted by $\sim$ or not
Proofs by Contradiction

Ctr: \[ \frac{\Gamma, A \vdash B \quad \Gamma, A \vdash \neg B}{\Gamma \vdash \neg A} \]

CtrN: \[ \frac{\Gamma, \neg A \vdash B \quad \Gamma, \neg A \vdash \neg B}{\Gamma \vdash A} \]
Double Negation

\[ \neg
\neg: \quad \frac{\Gamma \vdash A}{\Gamma \vdash \neg\neg A} \]

\[ \neg\neg i: \quad \frac{\Gamma \vdash A}{\Gamma \vdash \neg\neg A} \]

\[ \neg\neg e: \quad \frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A} \]

1. \( \Gamma \vdash A \) \quad \text{(premise)}
2. \( \Gamma, \neg A \vdash A \) \quad \text{(Ant 1)}
3. \( \Gamma, \neg A \vdash \neg A \) \quad \text{(Assm)}
4. \( \Gamma \vdash \neg\neg A \) \quad \text{(Ctr 2,3)}

1. \( \Gamma \vdash \neg\neg A \) \quad \text{(premise)}
2. \( \Gamma, \neg A \vdash \neg A \) \quad \text{(Assm)}
3. \( \Gamma, \neg A \vdash \neg\neg A \) \quad \text{(Ant 1)}
4. \( \Gamma \vdash A \) \quad \text{(CtrN 2,3)}
Double Negation

The proof of $\text{CtrN}$ by $\text{Ctr}$ and $\neg\neg\text{e}$:

$$
\text{CtrN: } \frac{\Gamma, \neg A \vdash B \quad \Gamma, \neg A \vdash \neg B}{\Gamma \vdash A}
$$

1. $\Gamma, \neg A \vdash B$ (premise)
2. $\Gamma, \neg A \vdash \neg B$ (premise)
3. $\Gamma \vdash \neg\neg A$ (Ctr 1,2)
4. $\Gamma \vdash A$ ($\neg\neg\text{e } 3$)
From a Contradiction Anything Follows

\[
\text{CtrA: } \quad \frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash B}
\]

1. \(\Gamma \vdash A\) (premise)
2. \(\Gamma \vdash \neg A\) (premise)
3. \(\Gamma, \neg B \vdash A\) (Ant 1)
4. \(\Gamma, \neg B \vdash \neg A\) (Ant 2)
5. \(\Gamma \vdash B\) (CtrN 3,4)
### Contraposition

Cp (a): \[
\begin{align*}
\Gamma, A & \vdash B \\
\Gamma, \neg B & \vdash \neg A
\end{align*}
\]

Cp (b): \[
\begin{align*}
\Gamma, A & \vdash \neg B \\
\Gamma, B & \vdash \neg A
\end{align*}
\]

Cp (c): \[
\begin{align*}
\Gamma, \neg A & \vdash B \\
\Gamma, \neg B & \vdash A
\end{align*}
\]

Cp (d): \[
\begin{align*}
\Gamma, \neg A & \vdash \neg B \\
\Gamma, B & \vdash A
\end{align*}
\]

1. \( \Gamma, A \vdash B \) (premise)
2. \( \Gamma, \neg B, A \vdash B \) (Ant 1)
3. \( \Gamma, \neg B, A \vdash \neg B \) (Assm)
4. \( \Gamma, \neg B \vdash \neg A \) (Ctr 2,3)

1. \( \Gamma, A \vdash \neg B \) (premise)
2. \( \Gamma, B, A \vdash B \) (Assm)
3. \( \Gamma, B, A \vdash \neg B \) (Ant 1)
4. \( \Gamma, B \vdash \neg A \) (Ctr 2,3)
Chaining of Arguments

\[
\text{Ch: } \frac{\Gamma \vdash A}{\Gamma, A \vdash B} \quad \frac{\Gamma, A \vdash B}{\Gamma \vdash B}
\]

1. $\Gamma \vdash A$ \hspace{2cm} (premise)
2. $\Gamma, A \vdash B$ \hspace{2cm} (premise)
3. $\Gamma, \neg B \vdash A$ \hspace{2cm} (Ant 1)
4. $\Gamma, \neg B \vdash \neg A$ \hspace{2cm} (Cp (a) 2)
5. $\Gamma \vdash B$ \hspace{2cm} (CtrN 3,4)
Proof by Cases

**PC:**

\[
\frac{\Gamma, A \vdash B \quad \Gamma, \neg A \vdash B}{\Gamma \vdash B}
\]

1. \(\Gamma, A \vdash B\) (premise)
2. \(\Gamma, \neg A \vdash B\) (premise)
3. \(\Gamma, \neg B \vdash A\) (Cp (c) 2)
4. \(\Gamma, \neg B \vdash \neg A\) (Cp (a) 1)
5. \(\Gamma \vdash B\) (CtrN 3,4)
Law of Excluded Middle (Tertium Non Datur)

\[ \vdash A \lor \neg A \]

1. \( A \vdash A \) (Assm)
2. \( A \vdash A \lor \neg A \) (\( \lor \)i 1)
3. \( \neg A \vdash \neg A \) (Assm)
4. \( \neg A \vdash A \lor \neg A \) (\( \lor \)i 2)
5. \( \vdash A \lor \neg A \) (PC 2,4)
Other Rules for Disjunction

\[\frac{\Gamma, \neg A \vdash B}{\Gamma \vdash A \lor B}\]

\[\frac{\Gamma, \neg B \vdash A}{\Gamma \vdash A \lor B}\]

1. \(\Gamma, A \vdash A\) (Assm)
2. \(\Gamma, A \vdash A \lor B\) (\(\lor_i 1\))
3. \(\Gamma, \neg A \vdash B\) (premise)
4. \(\Gamma, \neg A \vdash A \lor B\) (\(\lor_i 2\) 3)
5. \(\Gamma \vdash A \lor B\) (PC 2,4)
Implication: $A \rightarrow B$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$A \rightarrow B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Remark: also denoted by $\Rightarrow$ or $\supset$

\[ \rightarrow i: \frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \quad \rightarrow e: \frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \]

Remark: $\rightarrow e$ is better known as *modus ponens*
Examples of Proofs with Implication

\[ \frac{\Gamma \vdash A \rightarrow B}{\Gamma, A \vdash B} \]

Ch:

\[ \frac{\Gamma \vdash A, A \vdash B}{\Gamma \vdash B} \]

1. \( \Gamma \vdash A \rightarrow B \) (premise)
2. \( \Gamma, A \vdash A \rightarrow B \) (Ant 1)
3. \( \Gamma, A \vdash A \) (Assm)
4. \( \Gamma, A \vdash B \) (\( \rightarrow \)e 2,3)

1. \( \Gamma \vdash A \) (premise)
2. \( \Gamma, A \vdash B \) (premise)
3. \( \Gamma \vdash A \rightarrow B \) (\( \rightarrow \)i 2)
4. \( \Gamma \vdash B \) (\( \rightarrow \)e 3,1)
equivalece: \( A \leftrightarrow B \)

\[
\begin{array}{|c|c|c|}
\hline
A & B & A \leftrightarrow B \\
\hline
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\hline
\end{array}
\]

Remark: also denoted by \( \iff \) or \( \equiv \) or iff
Rules for Equivalence

↔i: \[ \frac{\Gamma, A \vdash B \quad \Gamma, B \vdash A}{\Gamma \vdash A \leftrightarrow B} \]

↔e₁: \[ \frac{\Gamma \vdash A \leftrightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \]

↔e₂: \[ \frac{\Gamma \vdash A \leftrightarrow B \quad \Gamma \vdash B}{\Gamma \vdash A} \]
Logical Constants: \( \bot \) (false), \( \top \) (true)

Remark: also denoted by 0, 1 or \texttt{ff}, \texttt{tt}

\[
\text{\( \bot \)e: } \frac{\Gamma \vdash \bot}{\Gamma \vdash A} \quad \quad \quad \text{\( \top \)i: } \frac{}{\vdash \top}
\]
\( \bot \) and Negation

1. \( \Gamma, A \vdash \bot \) (premise)
2. \( \Gamma, A \vdash \neg \bot \) (\( \bot \)e 1)
3. \( \Gamma \vdash \neg A \) (Ctr 1,2)

\[ \neg i: \quad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} \]

\[ \neg e: \quad \frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash \bot} \]
\( \bot \) and Negation

**Ctr:**

\[
\frac{\Gamma, A \vdash B \quad \Gamma, A \vdash \neg B}{\Gamma \vdash \neg A}
\]

1. \( \Gamma, A \vdash B \) (premise)
2. \( \Gamma, A \vdash \neg B \) (premise)
3. \( \Gamma, A \vdash \bot \) (\( \neg e \ 2,1 \))
4. \( \Gamma \vdash \neg A \) (\( \neg i \ 3 \))

**CtrA:**

\[
\frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash B}
\]

1. \( \Gamma \vdash A \) (premise)
2. \( \Gamma \vdash \neg A \) (premise)
3. \( \Gamma \vdash \bot \) (\( \neg e \ 2,1 \))
4. \( \Gamma \vdash B \) (\( \bot e \ 3 \))
If $\varphi$ is a well-formed formula, then also $\neg \varphi$ is a well-formed formula.

If $\varphi$ and $\psi$ are well-formed formulas, then also $(\varphi \land \psi)$, $(\varphi \lor \psi)$, $(\varphi \rightarrow \psi)$, and $(\varphi \leftrightarrow \psi)$ are well-formed formulas.

$\bot$ and $\top$ are well-formed formulas.

Abstract syntax tree.

Conventions for omitting parentheses.
Propositional Logic — Syntax

**At** — the set of atomic propositions

**Alphabet** — the set of symbols:
- atomic propositions: all elements from $At$
- logical connectives: $\neg$, $\land$, $\lor$, $\rightarrow$, $\leftrightarrow$, $\perp$, $\top$
- parentheses: $)$, $(

**Language** — which sequences of symbols are formulas:
- If $p \in At$, then $p$ is a formula.
- $\perp$ and $\top$ are formulas.
- If $\varphi$ is a formula, then also $\neg \varphi$ is a formula.
- If $\varphi$ and $\psi$ are formulas, then also $(\varphi \land \psi)$, $(\varphi \lor \psi)$, $(\varphi \rightarrow \psi)$, and $(\varphi \leftrightarrow \psi)$ are formulas.
- There are no other formulas than those created according to the previous rules.
Propositional Logic — Syntax

A more succinct description of the syntax using so called Backus-Naur form:

\[ \varphi ::= p \mid \bot \mid \top \mid \neg \varphi \mid (\varphi \land \varphi) \mid (\varphi \lor \varphi) \mid (\varphi \rightarrow \varphi) \mid (\varphi \leftrightarrow \varphi) \]

\( p \) represents an arbitrary atomic proposition from the set \( At \)

In the so called abstract syntax, we abstract from details such as parentheses, priorities, etc.:

\[ \varphi ::= p \mid \bot \mid \top \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi \]

An alternative notation for the abstract syntax:

\[ \varphi ::= p \mid \bot \mid \top \mid \neg \varphi_1 \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \lor \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \varphi_1 \leftrightarrow \varphi_2 \]
Propsitional Logic — Semantics

Truth valuation: \( \nu : \text{At} \rightarrow \{0, 1\} \)

- \( \nu \models \varphi \) — formula \( \varphi \) is true in valuation \( \nu \)
- \( \nu \not\models \varphi \) — formula \( \varphi \) is not true in valuation \( \nu \)

The definition of when a formula is true in valuation \( \nu \):

- \( \nu \models p \), where \( p \in \text{At} \), holds if and only if \( \nu(p) = 1 \).
- \( \nu \models \bot \) never holds (i.e., it always holds that \( \nu \not\models \bot \)).
- \( \nu \models \top \) always holds.
- \( \nu \models \neg \varphi \) holds if and only if \( \nu \not\models \varphi \).
- \( \nu \models \varphi \land \psi \) holds if and only if \( \nu \models \varphi \) and \( \nu \models \psi \).
- \( \nu \models \varphi \lor \psi \) holds if and only if \( \nu \models \varphi \) or \( \nu \models \psi \).
- \( \nu \models \varphi \rightarrow \psi \) holds if and only if \( \nu \not\models \varphi \) or \( \nu \models \psi \).
- \( \nu \models \varphi \leftrightarrow \psi \) holds if and only if \( \nu \models \varphi \) and \( \nu \models \psi \), or if \( \nu \not\models \varphi \) and \( \nu \not\models \psi \).
Propositional Logic — Semantics

\[ h_\neg : \{0, 1\} \rightarrow \{0, 1\} \]

<table>
<thead>
<tr>
<th>( x )</th>
<th>( h_\neg(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ h_\ast : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\} \quad \text{for } \ast \in \{\land, \lor, \rightarrow, \leftrightarrow\} \]

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( h_\land(x, y) )</th>
<th>( h_\lor(x, y) )</th>
<th>( h_\rightarrow(x, y) )</th>
<th>( h_\leftrightarrow(x, y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Propositional Logic — Semantics

\( \mathcal{L}_{At} \) — the set of all well-formed formulas

To each truth valuation \( \nu : At \rightarrow \{0, 1\} \), there is assigned a function

\[ \hat{\nu} : \mathcal{L}_{At} \rightarrow \{0, 1\} \]

- \( \hat{\nu}(p) = \nu(p) \) for \( p \in At \)
- \( \hat{\nu}(\bot) = 0 \)
- \( \hat{\nu}(\top) = 1 \)
- \( \hat{\nu}(\neg \varphi) = h_\neg(\hat{\nu}(\varphi)) \)
- \( \hat{\nu}(\varphi \ast \psi) = h_\ast(\hat{\nu}(\varphi), \hat{\nu}(\psi)) \) for \( \ast \in \{\land, \lor, \rightarrow, \leftrightarrow\} \)

- \( \nu \models \varphi \) — when \( \hat{\nu}(\varphi) = 1 \)
- \( \nu \not\models \varphi \) — when \( \hat{\nu}(\varphi) = 0 \)
Interpretations and Models

In general, when discussing a semantics of a logic, we talk about **interpretations**.

- In the case of propositional logic, an interpretation is just a truth valuation.

- For other logics, interpretations can be something different.

- If $\nu$ is an interpretation, the notation $\nu \models \varphi$ denotes that formula $\varphi$ is true in the interpretation $\nu$.

- An interpretation, where formula $\varphi$ holds, is called a **model** of the formula $\varphi$.

- When $\Gamma$ is a set of formulas, a **model** of this set of formulas is such interpretation that is a model of each formula from $\Gamma$. 
Logical Entailment

$\varphi$ – formula, $\Gamma$ – a set of formulas

Formula $\varphi$ **logically follows** from formulas $\Gamma$, which is written as

$$\Gamma \models \varphi,$$

if $\nu \models \varphi$ hold for each interpretation $\nu$, where for all formulas $\psi \in \Gamma$ it holds that $\nu \models \psi$.

**Remark:** The set $\Gamma$ can be infinite.

Instead of $\{\psi_1, \psi_2, \ldots, \psi_n\} \models \varphi$, we usually write $\psi_1, \psi_2, \ldots, \psi_n \models \varphi$.

Instead of $\emptyset \models \varphi$, we usually write $\models \varphi$. 
Tautologies, Contradictions, and Satisfiable Formulas

Formula $\varphi$ is:

- **tautology** — for every interpretation $\nu$ it holds that $\nu \models \varphi$

- **contradiction** — for every interpretation $\nu$ it holds that $\nu \not\models \varphi$

- **satisfiable** — there exists at least one interpretation, for which it holds that $\nu \models \varphi$

$\models \varphi$ — denotes that $\varphi$ is a tautology

**Remark:**

$$A_1, A_2, \ldots, A_n \models B$$

iff

$$\models A_1 \rightarrow (A_2 \rightarrow (\cdots \rightarrow (A_n \rightarrow B)\cdots))$$
Table Method

\[ p \land (q \rightarrow \neg r) \]

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Table Method

\(((A \rightarrow B) \rightarrow A) \rightarrow A\)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava)
Introd. to Theoretical Computer Science
February 6, 2014 45 / 401
**Deduction Systems**

**Deduction system** — a set of rules that determine, how proofs can look and what kinds of steps can be done in these proofs.

**Example:** A deduction system for propositional logic based on sequents (sequent calculus), containing the following rules:

<table>
<thead>
<tr>
<th>Assm</th>
<th>Ant</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\land i)</td>
<td>(\land e_1, \land e_2)</td>
</tr>
<tr>
<td>(\lor i_1, \lor i_2)</td>
<td>(\lor e)</td>
</tr>
<tr>
<td>(\to i)</td>
<td>(\to e)</td>
</tr>
<tr>
<td>(\leftrightarrow i)</td>
<td>(\leftrightarrow e_1, \leftrightarrow e_2)</td>
</tr>
<tr>
<td>(\neg i)</td>
<td>(\neg e)</td>
</tr>
<tr>
<td>(\top i)</td>
<td>(\bot e)</td>
</tr>
<tr>
<td>(\neg\neg e)</td>
<td></td>
</tr>
</tbody>
</table>

**Remark:** This is one of variants of so called **natural deduction**.
ϕ – a formula, Γ – a set of formulas (it can be infinite)

For a given deduction system \( \mathcal{D} \), the notation

\[ \Gamma \vdash_{\mathcal{D}} \phi \]

denotes that in the system \( \mathcal{D} \), there exists a proof of formula \( \phi \) from assumptions \( \Gamma \).

For a given deduction system \( \mathcal{D} \), it must specified precisely when \( \Gamma \vdash_{\mathcal{D}} \phi \) holds.

A system based on sequents (sequent calculus):

1. \( \Gamma \vdash_{\mathcal{D}} \phi \) holds iff there are some formulas \( \psi_1, \psi_2, \ldots, \psi_n \) from set \( \Gamma \) such that using the rules of the system it is possible to derive

\[ \psi_1, \psi_2, \ldots, \psi_n \vdash \phi \]

(i.e., there exists a proof of this sequent).
A deduction system $\mathcal{D}$ is:

- **sound** — if it holds that if $\Gamma \vdash_{\mathcal{D}} \varphi$ then $\Gamma \models \varphi$
  
  I.e., it is not possible to prove an incorrect conclusion in this system. What is derived, it really holds.

- **complete** — if it holds that if $\Gamma \models \varphi$ then $\Gamma \vdash_{\mathcal{D}} \varphi$
  
  I.e., everything that holds can be also proved in the system.
For proving that a given system is sound, it is sufficient to prove that every rule is sound.

**Example:** For proving that rule

\[
\land i: \quad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \land B}
\]

is sound, it is sufficient to show that:

- if $\Gamma \models A$ and $\Gamma \models B$, then $\Gamma \models A \land B$. 
It is possible to simulate the table method in the deduction system:

- The fact that $\varphi$ has truth value 1 in a given truth valuation, is represented by formula $\varphi$.

- The fact that $\varphi$ has truth value 0 in a given truth valuation, is represented by formula $\neg \varphi$. 
Table Method in the Deduction System

**Example:** Let us have atomic propositions $p, q, r$, and a formula $\varphi$ constructed from those atomic propositions using logical connectives.

- If for example $\nu_1 \models \varphi$ for valuation $\nu_1$, where
  
  $\nu_1(p) = 1$, $\nu_1(q) = 0$, $\nu_1(r) = 0$,

  it is possible to derive sequent
  
  $p, \neg q, \neg r \vdash \varphi$

- If for example $\nu_2 \not\models \varphi$ for valuation $\nu_2$, where
  
  $\nu_2(p) = 0$, $\nu_2(q) = 1$, $\nu_2(r) = 0$,

  it is possible to derive sequent
  
  $\neg p, q, \neg r \vdash \neg \varphi$
Table Method in the Deduction System

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ∧ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

For example, it is possible to derive the following rules for conjunction:

\[
\begin{align*}
\Gamma &\vdash \neg A & \Gamma &\vdash \neg B \\
\hline
\Gamma &\vdash \neg(A \land B) \\
\end{align*}
\]

\[
\begin{align*}
\Gamma &\vdash \neg A & \Gamma &\vdash B \\
\hline
\Gamma &\vdash \neg(A \land B) \\
\end{align*}
\]

\[
\begin{align*}
\Gamma &\vdash A & \Gamma &\vdash \neg B \\
\hline
\Gamma &\vdash \neg(A \land B) \\
\end{align*}
\]

\[
\begin{align*}
\Gamma &\vdash A & \Gamma &\vdash B \\
\hline
\Gamma &\vdash A \land B \\
\end{align*}
\]
The table method can be viewed as an example of a proof by cases:

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
<th>$\varphi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

1. $\neg p_1, \neg p_2, \neg p_3 \vdash \varphi$ (PC 2,1)
2. $\neg p_1, \neg p_2, p_3 \vdash \varphi$ (PC 4,3)
3. $\neg p_1, p_2, \neg p_3 \vdash \varphi$ (PC 6,5)
4. $\neg p_1, p_2, p_3 \vdash \varphi$ (PC 8,7)
5. $p_1, \neg p_2, \neg p_3 \vdash \varphi$ (PC 10,9)
6. $p_1, \neg p_2, p_3 \vdash \varphi$ (PC 12,11)
7. $p_1, p_2, \neg p_3 \vdash \varphi$ (PC 14,13)
Equivalence of Formulas

Formulas $\varphi$ and $\psi$ are **logically equivalent**, which is denoted $\varphi \leftrightarrow \psi$, if for every interpretation $\nu$ it holds

$$\nu \models \varphi \quad \text{iff} \quad \nu \models \psi.$$

**Remark:** $\varphi \leftrightarrow \psi$ holds iff $\varphi \models \psi$ and $\psi \models \varphi$.

Formulas $\varphi$ and $\psi$ are **provably equivalent** in a given deduction system, which is denoted $\varphi \vdash \psi$, if

$$\varphi \vdash \psi \quad \text{and} \quad \psi \vdash \varphi.$$

**Remark:** If the deduction system is sound, then from $\varphi \vdash \psi$ follows $\varphi \leftrightarrow \psi$, if it is complete, then from $\varphi \leftrightarrow \psi$ follows $\varphi \vdash \psi$. 
Equivalence of Formulas

\[ \varphi \iff \psi \iff \models \varphi \iff \psi. \]
\[ \varphi \dashv \vdash \psi \iff \vdash \varphi \iff \psi. \]

For arbitrary formulas \( \varphi \), \( \psi \), and \( \chi \), it holds:

- \( \varphi \iff \varphi \).
- If \( \varphi \iff \psi \), then \( \psi \iff \varphi \).
- If \( \varphi \iff \psi \) and \( \psi \iff \chi \), then \( \varphi \iff \chi \).

Similarly:

- \( \varphi \dashv \vdash \varphi \).
- If \( \varphi \dashv \vdash \psi \), then \( \psi \dashv \vdash \varphi \).
- If \( \varphi \dashv \vdash \psi \) and \( \psi \dashv \vdash \chi \), then \( \varphi \dashv \vdash \chi \).
Equivalence of Formulas

When $A \vdash A'$ holds:

- It is possible to prove $\Gamma \vdash A'$ from $\Gamma \vdash A$ (and vice versa).
- It is possible to prove $\Gamma, A', \Delta \vdash B$ from $\Gamma, A, \Delta \vdash B$ (and vice versa).

When $A \vdash A'$ then it also holds that:

- $\neg A \vdash \neg A'$
- $A \land B \vdash A' \land B$, $B \land A \vdash B \land A'$
- $A \lor B \vdash A' \lor B$, $B \lor A \vdash B \lor A'$
- $A \rightarrow B \vdash A' \rightarrow B$, $B \rightarrow A \vdash B \rightarrow A'$
- $A \leftrightarrow B \vdash A' \leftrightarrow B$, $B \leftrightarrow A \vdash B \leftrightarrow A'$

(similarly for $\iff$)
Some Important Equivalences

Remark: All equivalences mentioned below are also provable ($\vdash\vdash$).

$$A \iff \neg\neg A,$$

Associativity of $\land$, $\lor$, $\leftrightarrow$:

$$(A \land B) \land C \iff A \land (B \land C)$$
$$(A \lor B) \lor C \iff A \lor (B \lor C)$$
$$(A \leftrightarrow B) \leftrightarrow C \iff A \leftrightarrow (B \leftrightarrow C)$$

Commutativity of $\land$, $\lor$, $\leftrightarrow$:

$$A \land B \iff B \land A \quad A \lor B \iff B \lor A \quad A \leftrightarrow B \iff B \leftrightarrow A$$

Idempotence of $\land$ and $\lor$:

$$A \land A \iff A \quad A \lor A \iff A$$
Some Important Equivalences

Distributive laws for $\land$ and $\lor$:

$$A \land (B \lor C) \iff (A \land B) \lor (A \land C) \quad A \lor (B \land C) \iff (A \lor B) \land (A \lor C)$$

De Morgan’s laws:

$$\neg(A \land B) \iff \neg A \lor \neg B \quad \neg(A \lor B) \iff \neg A \land \neg B$$

Conjunction, resp. disjunction, of formulas $A$ and $\neg A$ is equivalent to $\bot$, resp. $\top$:

$$A \land \neg A \iff \bot \quad A \lor \neg A \iff \top$$
Some Important Equivalences

Equivalences involving $\bot$ and $\top$:

\[
\begin{align*}
\neg \bot & \iff \top \\
A \land \bot & \iff \bot \\
A \lor \bot & \iff A \\
A \rightarrow \bot & \iff \neg A \\
\bot \rightarrow A & \iff \top \\
A \leftrightarrow \bot & \iff \neg A \\
\neg \top & \iff \bot \\
A \land \top & \iff A \\
A \lor \top & \iff \top \\
A \rightarrow \top & \iff \top \\
\top \rightarrow A & \iff A \\
A \leftrightarrow \top & \iff A
\end{align*}
\]

Equivalences for replacing the logical connective implication:

\[
A \rightarrow B \iff \neg A \lor B \quad \neg (A \rightarrow B) \iff A \land \neg B
\]

Equivalences for replacing the logical connective equivalence:

\[
\begin{align*}
A \leftrightarrow B & \iff (A \rightarrow B) \land (B \rightarrow A) \\
A \leftrightarrow B & \iff (A \land B) \lor (\neg A \land \neg B) \\
A \leftrightarrow B & \iff (A \lor \neg B) \land (\neg A \lor B)
\end{align*}
\]
Equivalent Transformations

Instead of $A_1 \iff A_2, A_2 \iff A_3, \ldots, A_{n-1} \iff A_n$, it is possible to write

$$A_1 \iff A_2 \iff A_3 \iff \cdots \iff A_{n-1} \iff A_n$$

For arbitrary $A_i, A_j$, where $i, j \in \{1, 2, \ldots, n\}$, it holds that $A_i \iff A_j$.

Similarly, instead of $A_1 \vdash A_2, A_2 \vdash A_3, \ldots, A_{n-1} \vdash A_n$, it is possible to write

$$A_1 \vdash A_2 \vdash A_3 \vdash \cdots \vdash A_{n-1} \vdash A_n$$

For arbitrary $A_i, A_j$, where $i, j \in \{1, 2, \ldots, n\}$, it holds that $A_i \vdash A_j$. 
Let us consider only formulas of propositional logic.

- **Literal** — an atomic proposition or its negation, e.g., $p$ or $\neg r$
  
  $$L ::= p \mid \neg p$$

- **Elementary conjunction** — a conjunction of one of more literals, e.g., $p \land \neg q$, $r$, $q \land \neg r \land p$
  
  $$C ::= L \mid L \land C$$

- **Elementary disjunction** (clause) — a disjunction of one or more literals, e.g., $p \lor \neg q$, $r$, $q \lor \neg r \lor p$
  
  $$D ::= L \mid L \lor D$$
Normal Forms of Formulas

- **Disjunctive normal form (DNF)** — a disjunction of one or more elementary conjunctions, e.g., \((p \land \neg q) \lor (\neg r) \lor (\neg r \land \neg p \land \neg q)\)

  \[ E ::= C \mid C \lor E \]

- **Conjunctive normal form (CNF)** — a conjunction of one or more elementary disjunctions (clauses), e.g., \((p \lor \neg q) \land (\neg r) \land (\neg r \lor \neg p \lor \neg q)\)

  \[ F ::= D \mid D \land F \]

**Remark:** Formulas \(\bot\) and \(\top\) will be also considered to be in DNF and CNF.
For formulas in CNF, it is easy to determine if they are tautologies or not:

In a tautology in CNF, every clause contains as literals some atomic proposition $p$ and its negation $\neg p$.

For formulas in DNF, it is easy to determine if they are contradictions or not:

In a contradiction in DNF, every elementary conjunction contains as literals some atomic proposition $p$ and its negation $\neg p$. 

Normal Forms of Formulas

<table>
<thead>
<tr>
<th></th>
<th>p</th>
<th>q</th>
<th>r</th>
<th>ϕ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

DNF:
$$(\neg p \land \neg q \land r) \lor (\neg p \land q \land \neg r) \lor (p \land \neg q \land r)$$

KNF:
$$(p \lor q \lor r) \land (p \lor \neg q \lor \neg r) \land (\neg p \lor q \lor r) \land (\neg p \lor \neg q \lor r) \land (\neg p \lor \neg q \lor \neg r)$$
Normal Forms of Formulas

When we consider a fixed finite set of atomic propositions $At$:

- **Complete disjunctive normal form (CDNF)** — a formula in DNF, where every elementary conjunction contains every atomic proposition from $At$ exactly once.

  **Example:** $(p \land \neg q \land \neg r) \lor (p \land q \land \neg r) \lor (\neg p \land q \land \neg r)$

- **Complete conjunctive normal form (CCNF)** — a formula in CNF, where every clause contains every atomic proposition from $At$ exactly once.

  **Example:** $(p \lor \neg q \lor \neg r) \land (p \lor q \lor \neg r) \land (\neg p \lor q \lor \neg r)$

**Remark:** In the examples is $At = \{p, q, r\}$. 
**Set** — a collection of elements

Element $a$ is a member of set $S$:

$$a \in S$$

Element $a$ is not a member of set $S$:

$$a \notin S$$

A finite set can be expressed by a list of elements of the set:

$$S = \{a, b, c\}$$

**Subset:** $S \subseteq T$ — every element of set $S$ is a member of set $T$
Ordered \( n \)-tuples

**Ordered \( n \)-tuple:** \((a_1, a_2, \ldots, a_n)\) or \(\langle a_1, a_2, \ldots, a_n \rangle\)

- **Ordered pair:** \((a, b)\) or \(\langle a, b \rangle\)
- **Ordered triple:** \((a, b, c)\) or \(\langle a, b, c \rangle\)
- \(\ldots\)

**Cartesian product:**

\[ S_1 \times S_2 \times \cdots \times S_n \]

— the set of all ordered \( n \)-tuples

\[(a_1, a_2, \ldots, a_n)\]

where \(a_1 \in S_1, a_2 \in S_2, \ldots, a_n \in S_n\)
Relation: \( R \subseteq S_1 \times S_2 \times \cdots \times S_n \)

- to express relationships between \( n \)-tuples of elements
- \( n \) — the arity of the relation
  - \( n = 1 \) — unary
  - \( n = 2 \) — binary
  - \( n = 3 \) — ternary

Example: Binary relation \( R_1 \subseteq \mathbb{N} \times \mathbb{N} \) consisting of pairs of numbers \((m, n)\), where \( m < n \)

\[
(m, n) \in R_1 \iff m < n
\]

Remark: \( \mathbb{N} = \{0, 1, 2, \ldots\} \)
Predicate — assigns a truth value to a given $n$-tuple of elements $(a_1, a_2, \ldots, a_n)$, depending on whether this $n$-tuple is or is not in a given relation $R$: 

$$R(a_1, a_2, \ldots, a_n)$$

I.e., $R(a_1, a_2, \ldots, a_n)$ holds iff $(a_1, a_2, \ldots, a_n) \in R$.

Example: Predicate $R_1$, where 

$$R_1(m, n)$$

holds iff $(m, n) \in R_1$, i.e., when $m < n$
Unary relation — $R \subseteq S$ represents some property of elements from set $S$

Example: Unary relation $R_2 \subseteq \mathbb{N}$ consisting of those numbers that are primes

Predicate $R_2$ expressing that $n$ is a prime

\[ R_2(n) \]
functions

Function — a binary relations \( f \subseteq S \times T \) satisfying:

- if \((x, y_1) \in f\) and \((x, y_2) \in f\), then \(y_1 = y_2\)

I.e., for every element \(x \in S\), there exists at most one element \(y \in T\) such that \((x, y) \in f\).

This element is denoted \(f(x)\)

Function \(f\) — represents a mapping that assigns elements from set \(T\) to elements from set \(S\):

\[ f : S \rightarrow T \]

Total vs. parcial functions
Function \( f : S \to T \), where \( S = A_1 \times A_2 \times \cdots \times A_n \):

\[
f : A_1 \times A_2 \times \cdots \times A_n \to T
\]

\( n \) — the arity of function \( f \)

Instead of \( f((a_1, a_2, \ldots, a_n)) \), we write \( f(a_1, a_2, \ldots, a_n) \).

**Example:** Binary function \( f_1 : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \), which assigns to a pair of numbers their sum

\[
f_1(x, y) = x + y
\]
**Structures**

**Structure** — a non-empty set of elements together with several relations and functions over the elements of this set

**Example:** Set $\mathbb{N} = \{0, 1, 2, \ldots\}$ together with the following relations and functions:

- **Unary function** $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(x) = x + 1$
- **Binary function** $g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, where $g(x, y) = x + y$
- **Binary function** $h : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, where $h(x, y) = x \cdot y$
- **Binary relation** $P \subseteq \mathbb{N} \times \mathbb{N}$, where $(x, y) \in P$ iff $x = y$
- **Binary relation** $Q \subseteq \mathbb{N} \times \mathbb{N}$, where $(x, y) \in Q$ iff $x < y$
**Example:** Set $A = \{a, b, c\}$ together with the following functions $f$ and $g$ and relation $R$:

- **unary function** $f : A \rightarrow A$ and binary function $g : A \times A \rightarrow A$

<table>
<thead>
<tr>
<th>$x$</th>
<th>$f(x)$</th>
<th>$g$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
<td></td>
<td>$a$</td>
<td>$c$</td>
<td>$a$</td>
</tr>
<tr>
<td>$b$</td>
<td>$a$</td>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>$c$</td>
<td>$b$</td>
<td></td>
<td>$c$</td>
<td>$a$</td>
<td>$c$</td>
</tr>
</tbody>
</table>

- **binary relation** $R \subseteq A \times A$, where

$$R = \{(a, a), (a, c), (b, b), (c, a), (c, b)\}$$
**Signatures**

**Signature** — tuple

\((\mathcal{P}, \mathcal{F}, \mathcal{C}, \text{arity})\)

- \(\mathcal{P}\) — a set of **predicate symbols**
- \(\mathcal{F}\) — a set of **function symbols**
- \(\mathcal{C}\) — a set of **constant symbols**
- \(\text{arity}: \mathcal{P} \cup \mathcal{F} \to \mathbb{N}\) — a function determining the **arity** of each predicate or function symbol

**Example:** Signature \(S = (\mathcal{P}, \mathcal{F}, \mathcal{C}, \text{arity})\), where \(\mathcal{P} = \{P, Q, R\}\), \(\mathcal{F} = \{f, g\}\), \(\mathcal{C} = \{c, d\}\), and where \(\text{arity}(P) = 1\), \(\text{arity}(Q) = 1\), \(\text{arity}(R) = 2\), \(\text{arity}(f) = 2\), \(\text{arity}(g) = 1\)
Interpretations

Signature $S = (\mathcal{P}, \mathcal{F}, \mathcal{C}, \text{arity})$

**Interpretation**: $\mathcal{U} = (A, a)$

- $A$ — **universe** — an arbitrary non-empty set
- $a$ — a mapping assigning meaning to every symbol in signature $S$:
  - for $P \in \mathcal{P}$ (where $\text{arity}(P) = n$):
    $a(P) = P^2$, where $P^2 \subseteq A^n$ is an arbitrary $n$-ary relation over set $A$
  - for $f \in \mathcal{F}$ (where $\text{arity}(f) = n$):
    $a(f) = f^2$, where $f^2 : A^n \rightarrow A$ is an arbitrary $n$-ary (total) function over set $A$
  - for $c \in \mathcal{C}$:
    $a(c) = c^2$, where $c^2 \in A$ is an arbitrary element of the universe $A$
**Valuations**

**Variables:** \( Var = \{x_0, x_1, x_2, \ldots \} \)

**Remark:** Variables will be denoted \( x, y, z \)

Assume an interpretation \( \mathcal{A} = (A, a) \).

**Valuation** — determines values of variables, it assigns elements of the universe to variables:

\[
v : Var \rightarrow A
\]

An interpretation and a valuation:

\[
\mathcal{I} = (\mathcal{A}, v)
\]
Terms

**Term** — an expression denoting an element of the universe:

- $x$ — where $x \in Var$
- $c$ — where $c \in C$
- $f(t_1, t_2, \ldots, t_n)$ — where $f \in F$, $\text{arity}(f) = n$, and where $t_1, t_2, \ldots, t_n$ are terms

Examples of terms: $x$ $c$ $f(x, y)$ $f(g(x), f(c, y))$

Syntax:

$$t ::= x \mid c \mid f(t, t, \ldots, t)$$
Terms

An interpretation and a valuation $\mathcal{I} = (\mathcal{A}, v)$

$\mathcal{I}(t)$ — a value of term $t$ in interpretation $\mathcal{A}$ a valuation $v$

- for $x \in \text{Var}$: $\mathcal{I}(x) = v(x)$
- for $c \in \mathcal{C}$: $\mathcal{I}(c) = c^\mathcal{A}$
- for $f \in \mathcal{F}$ (where $\text{arity}(f) = n$) and terms $t_1, t_2, \ldots, t_n$:
  $\mathcal{I}(f(t_1, t_2, \ldots, t_n)) = f^{\mathcal{A}}(\mathcal{I}(t_1), \mathcal{I}(t_2), \ldots, \mathcal{I}(t_n))$
Atomic Formulas

**Atomic formula:**

\[ P(t_1, t_2, \ldots, t_n) \]

where \( P \in \mathcal{P} \) (where \( \text{arity}(P) = n \)) and \( t_1, t_2, \ldots, t_n \) are terms

Examples of atomic formulas: \( P(x) \quad Q(g(g(c))) \quad R(f(x, y), x) \)

An interpretation and a valuation \( \mathcal{I} = (\mathcal{A}, \nu) \)

Formula \( \varphi \) is true in interpretation \( \mathcal{A} \) and valuation \( \nu \):

\[ \mathcal{I} \models \varphi \]

\( \mathcal{I} \models P(t_1, t_2, \ldots, t_n) \) iff \( (\mathcal{I}(t_1), \mathcal{I}(t_2), \ldots, \mathcal{I}(t_n)) \in P^\mathcal{A} \).
Equality (identity): denoted by $=\ $\n
Atomic formula:

$$t_1 = t_2$$

Examples of atomic formulas with equality:

$$x = y \quad f(f(x, y), z) = g(x)$$

- $\mathcal{I} \models t_1 = t_2$ iff $\mathcal{I}(t_1) = \mathcal{I}(t_2)$
It is possible to create formulas from smaller subformulas using logical connectives: $\neg$, $\land$, $\lor$, $\rightarrow$, $\leftrightarrow$, $\bot$, $\top$.

It is also possible to use **quantifiers**:

- universal quantifier: $\forall$
- existential quantifier: $\exists$

If $\varphi$ is a formula and $x$ is a variable ($x \in \text{Var}$), then also:

- $\forall x \varphi$ is a formula
  — it represents propositions “for every element $x$, $\varphi$ holds”, “for each $x$, $\varphi$ holds”, etc.

- $\exists x \varphi$ is a formula
  — it represents propositions “there exists an element $x$, for which $\varphi$ holds”, “for some $x$, $\varphi$ holds”, etc.
Interpretation $\mathcal{A} = (A, a)$, valuation $v : \text{Var} \rightarrow A$

$x \in \text{Var}, a \in A$

Notation

$$v[x \mapsto a]$$

denotes the valuation $v' : \text{Var} \rightarrow A$, which assignes to every variable the value as valuation $v$, except that it assignes value $a$ to variable $x$

I.e., for $y \in \text{Var}$ is

$$v'(y) = \begin{cases} a & \text{if } y = x \\ v(y) & \text{otherwise} \end{cases}$$

$\mathcal{I} = (\mathcal{A}, v)$

$$\mathcal{I}[x \mapsto a] = (\mathcal{A}, v[x \mapsto a])$$
Quantifiers

- $\mathcal{I} \models \forall x \varphi$ holds iff for every $a \in A$ it holds that $\mathcal{I}[x \mapsto a] \models \varphi$
- $\mathcal{I} \models \exists x \varphi$ holds iff there exists some $a \in A$ such that $\mathcal{I}[x \mapsto a] \models \varphi$

Examples of formulas:
- $\forall x \exists y R(x, y)$ — for each $x$ there exists some $y$ such that $x$ and $y$ are in relation $R$
- $\neg \exists x (P(x) \land Q(x))$ — there does not exist such $x$, for which both $P(x)$ and $Q(x)$ would be true
- $\exists x P(x) \rightarrow \forall y Q(y)$ — if there is some $x$, for which $P(x)$ holds, then for every $y$, $Q(y)$ holds
First order predicate logic — summary

Symbols:

- **Logical symbols:**
  - **variables:** $x \in Var$, where $Var = \{x_0, x_1, x_2, \ldots\}$
  - **logical connectives:** $\neg, \land, \lor, \rightarrow, \leftrightarrow$
  - **quantifiers:** $\forall, \exists$
  - **parentheses:** $), (\,$
  - **equality:** $=$

- **Non-logical symbols — given by signature $S = (P, F, C, \text{arity})$:**
  - **predicate symbols:** $P \in P$
  - **function symbols:** $f \in F$
  - **constant symbols:** $c \in C$

Syntax:

$$
t ::= x \mid c \mid f(t, t, \ldots, t)
$$

$$
\varphi ::= P(t, t, \ldots, t) \mid t = t \mid \bot \mid \top \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi
$$

$$
\mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi \mid \forall x \varphi \mid \exists x \varphi
$$
First order predicate logic — summary

Semantics:

A value of a term for $\mathcal{I} = (\mathcal{A}, \nu)$:

- for $x \in \text{Var}$: $\mathcal{I}(x) = \nu(x)$
- for $c \in \mathcal{C}$: $\mathcal{I}(c) = c^\mathcal{I}$
- for $f \in \mathcal{F}$ (where $\text{arity}(f) = n$) and terms $t_1, t_2, \ldots, t_n$:
  $\mathcal{I}(f(t_1, t_2, \ldots, t_n)) = f^\mathcal{I}(\mathcal{I}(t_1), \mathcal{I}(t_2), \ldots, \mathcal{I}(t_n))$
A truth value of a formula for $\mathcal{I} = (\mathcal{A}, a)$:

- For $P \in \mathcal{P}$, where $\text{arity}(P) = n$, and for terms $t_1, t_2, \ldots, t_n$, $\mathcal{I} \models P(t_1, t_2, \ldots, t_n)$ holds iff $(\mathcal{I}(t_1), \mathcal{I}(t_2), \ldots, \mathcal{I}(t_n)) \in P^a$.

- For terms $t_1, t_2$, $\mathcal{I} \models t_1 = t_2$ holds iff $\mathcal{I}(t_1) = \mathcal{I}(t_2)$.

- $\mathcal{I} \models \bot$ never holds, i.e., $\mathcal{I} \not\models \bot$ always holds.

- $\mathcal{I} \models \top$ always holds.

- $\mathcal{I} \models \neg \varphi$ holds iff $\mathcal{I} \not\models \varphi$.

- $\mathcal{I} \models \varphi \land \psi$ holds iff $\mathcal{I} \models \varphi$ and $\mathcal{I} \models \psi$.

- $\mathcal{I} \models \varphi \lor \psi$ holds iff $\mathcal{I} \models \varphi$ or $\mathcal{I} \models \psi$.

- $\mathcal{I} \models \varphi \rightarrow \psi$ holds iff $\mathcal{I} \not\models \varphi$ or $\mathcal{I} \models \psi$.

- $\mathcal{I} \models \varphi \leftrightarrow \psi$ holds iff $\mathcal{I} \models \varphi$ and $\mathcal{I} \models \psi$, or if $\mathcal{I} \not\models \varphi$ and $\mathcal{I} \not\models \psi$.

- $\mathcal{I} \models \forall x \varphi$ holds iff for every $a \in A$ it holds that $\mathcal{I}[x \mapsto a] \models \varphi$.

- $\mathcal{I} \models \exists x \varphi$ holds iff there exists some $a \in A$ such that $\mathcal{I}[x \mapsto a] \models \varphi$. 
Every occurrence of variable $x$ in a subformula of the form $\exists x \varphi$ or $\forall x \varphi$ is bound.

An occurrence of a variable, which is not bound, is free.

free$(t)$ — the set of variables, which occur as free variables in term $t$:
- $\text{free}(x) = \{x\}$ for $x \in \text{Var}$
- $\text{free}(c) = \emptyset$ for $c \in C$
- $\text{free}(f(t_1, t_2, \ldots, t_n)) = \text{free}(t_1) \cup \text{free}(t_2) \cup \ldots \cup \text{free}(t_n)$ for $f \in F$

free$(\varphi)$ — the set variables, which occur as free variables in formula $\varphi$:
- $\text{free}(P(t_1, t_2, \ldots, t_n)) = \text{free}(t_1) \cup \text{free}(t_2) \cup \ldots \cup \text{free}(t_n)$ for $P \in P$
- $\text{free}(t_1 = t_2) = \text{free}(t_1) \cup \text{free}(t_2)$
- $\text{free}(\bot) = \text{free}(\top) = \emptyset$
- $\text{free}(\neg \varphi) = \text{free}(\varphi)$
- $\text{free}(\varphi \ast \psi) = \text{free}(\varphi) \cup \text{free}(\psi)$ for $\ast \in \{\land, \lor, \to, \leftrightarrow\}$
- $\text{free}(Qx \varphi) = \text{free}(\varphi) - \{x\}$ for $Q \in \{\exists, \forall\}$ and $x \in \text{Var}$
The truth value of formula $\varphi$ for $I = (A, v)$ depends only on $A$ a values assigned by $v$ to variables from the set $\text{free}(\varphi)$.

In particular, in the case when $\text{free}(\varphi) = \emptyset$, the truth value of $\varphi$ in $I = (A, v)$ depends only on the interpretation $A$.

For formulas $\varphi$, where $\text{free}(\varphi) = \emptyset$, we can write

$$A \models \varphi \quad \text{or} \quad A \not\models \varphi$$

Formula $\varphi$, where $\text{free}(\varphi) = \emptyset$, is called a closed formula or a sentence.
Substitution

$\varphi$ – formula, $x$ – variable, $t$ – term

The formula obtained when we substitute term $t$ for free occurrences of variable $x$ in formula $\varphi$:

$\varphi[t/x]$

Example: $\varphi := \forall x(P(x) \rightarrow R(f(x, z), y))$, $t := g(f(y, w))$

Formula $\varphi[t/z]$: 

$$\forall x(P(x) \rightarrow R(f(x, g(f(y, w))), y))$$

Remark: It is necessary to avoid the situation when a free occurrence of some variable in term $t$ becomes bound after the substitution. In such case, it is necessary to rename the bound variable.
Deduction System for First Order Predicate Logic

A deduction system consisting of the same rules as the system for propositional logic.

Some additional rules for quantifiers and equality.

\[ \forall e: \frac{\Gamma \vdash \forall x A}{\Gamma \vdash A[t/x]} \]

Example:

\[ \forall x (P(x) \rightarrow Q(x)), \ P(c) \vdash Q(c) \]

1. \( \Gamma \vdash \forall x (P(x) \rightarrow Q(x)) \) (Assm)
2. \( \Gamma \vdash P(c) \rightarrow Q(c) \) (\( \forall e 1 \))
3. \( \Gamma \vdash P(c) \) (Assm)
4. \( \Gamma \vdash Q(c) \) (\( \rightarrow e 2,3 \))

\[ \Gamma := \forall x (P(x) \rightarrow Q(x)), \ P(c) \]
Universal Quantifier

\[ \forall i: \quad \frac{\Gamma \vdash A[y/x]}{\Gamma \vdash \forall x A} (y \not\in \text{free}(\Gamma, \forall x A)) \]

A special case:

\[ \frac{\Gamma \vdash A}{\Gamma \vdash \forall x A} (x \not\in \text{free}(\Gamma)) \]
Universal Quantifier

Example: \( \forall x (P(x) \lor Q(x)), \forall x (Q(x) \rightarrow R(x)) \vdash \forall x (P(x) \lor R(x)) \)

1. \( \Gamma \vdash \forall x (P(x) \lor Q(x)) \) (Assm)
2. \( \Gamma \vdash P(x) \lor Q(x) \) (\( \forall e \, 1 \))
3. \( \Gamma, P(x) \vdash P(x) \) (Assm)
4. \( \Gamma, P(x) \vdash P(x) \lor R(x) \) (\( \lor i_1 \, 3 \))
5. \( \Gamma, Q(x) \vdash \forall x (Q(x) \rightarrow R(x)) \) (Assm)
6. \( \Gamma, Q(x) \vdash Q(x) \rightarrow R(x) \) (\( \forall e \, 5 \))
7. \( \Gamma, Q(x) \vdash Q(x) \) (Assm)
8. \( \Gamma, Q(x) \vdash R(x) \) (\( \rightarrow e \, 6,7 \))
9. \( \Gamma, Q(x) \vdash P(x) \lor R(x) \) (\( \lor i_2 \, 8 \))
10. \( \Gamma \vdash P(x) \lor R(x) \) (\( \forall e \, 2,4,9 \))
11. \( \Gamma \vdash \forall x (P(x) \lor R(x)) \) (\( \forall i \, 10 \))

\( \Gamma := \forall x (P(x) \lor Q(x)), \forall x (Q(x) \rightarrow R(x)) \)
Existential Quantifier

\[ \exists i : \Gamma \vdash A[t/x] \]

\[ \Gamma \vdash \exists x A \]

\[ \exists e : \frac{\Gamma \vdash \exists x A \quad \Gamma, A[y/x] \vdash B}{\Gamma \vdash B} (y \not\in \text{free}(\Gamma, \exists x A, B)) \]

A special case:

\[ \frac{\Gamma \vdash \exists x A \quad \Gamma, A \vdash B}{\Gamma \vdash B} (x \not\in \text{free}(\Gamma, B)) \]
Equality

\[ =_i: \quad \vdash t = t \]

\[ =_e: \quad \Gamma \vdash t = t' \quad \Gamma \vdash A[t/x] \]
\[ \quad \Gamma \vdash A[t'/x] \]

Example:

\[ \Gamma \vdash t_1 = t_2 \]
\[ \quad \vdash t_1 = t_1 \]
\[ \quad \Gamma \vdash t_1 = t_1 \quad \text{(Ant 2)} \]
\[ \Gamma \vdash (x = t_1)[t_1/x] \quad \text{(rep. 3, } x \notin \text{ free}(t_1)) \]
\[ \Gamma \vdash (x = t_1)[t_2/x] \quad \text{(}= e \ 1,4) \]
\[ \Gamma \vdash t_2 = t_1 \quad \text{(rep. 5)} \]
Remark: All following equivalences are also provable ($\vdash \vdash$)

$$
\neg \forall x \varphi \iff \exists x \neg \varphi \\
\neg \exists x \varphi \iff \forall x \neg \varphi
$$

When $x \not\in \text{free}(\psi)$:

$$
(\forall x \varphi) \land \psi \iff \forall x (\varphi \land \psi) \\
(\forall x \varphi) \lor \psi \iff \forall x (\varphi \lor \psi) \\
(\exists x \varphi) \land \psi \iff \exists x (\varphi \land \psi) \\
(\exists x \varphi) \lor \psi \iff \exists x (\varphi \lor \psi)
$$
Some Important Equivalences

\[(\forall x \varphi) \land (\forall x \psi) \iff \forall x (\varphi \land \psi)\]
\[(\exists x \varphi) \lor (\exists x \psi) \iff \exists x (\varphi \lor \psi)\]

\[\forall x \forall y \varphi \iff \forall y \forall x \varphi\]
\[\exists x \exists y \varphi \iff \exists y \exists x \varphi\]

When \( y \notin \text{free}(\varphi) \):

\[\forall x \varphi \iff \forall y (\varphi[y/x])\]
\[\exists x \varphi \iff \exists y (\varphi[y/x])\]

When \( x \notin \text{free}(\varphi) \):

\[\forall x \varphi \iff \varphi\]
\[\exists x \varphi \iff \varphi\]
There Exists Exactly One

Let us assume that:

- $\varphi$ is a formula
- variable $y$ is a different variable than $x$
- $y \in \text{free}(\varphi)$

Proposition “there is exactly one element $x$, for which $\varphi$ holds” can be represented by formula

$$\exists x(\varphi \land \forall y(\varphi[y/x] \rightarrow x = y))$$

It can be denoted also:

$$\exists_{=1} x \varphi$$

Remark: With the same meaning as $\exists_{=1}$, the following symbols are also used: $\exists_1$, $\exists^=1$, $\exists!$. 

Signature (\{<\}, \{\sigma, +, \cdot\}, \{0\}, \text{arity}), where \text{arity}(<) = 2, \text{arity}(\sigma) = 1, \text{arity}(+) = 2, \text{arity}(\cdot) = 2

Examples of Formulas:

- \( \forall x \exists y (x < y) \) — for each natural number, there exists a greater natural number
- \( \sigma(0) + \sigma(0) = \sigma(\sigma(0)) \) — it holds that \( 1 + 1 = 2 \)
- \( \sigma(0) < x \land \neg \exists y \exists z (\sigma(0) < y \land \sigma(0) < z \land y \cdot z = x) \) — formula with free variable \( x \), representing proposition that \( x \) is a prime number
Natural Numbers — Robinson and Peano Arithmetic

Axioms:

- $\forall x (\sigma(x) \neq 0)$
- $\forall x \forall y (\sigma(x) = \sigma(y) \rightarrow x = y)$
- $\forall x (x = 0 \lor \exists y (\sigma(y) = x))$
- $\forall x (x + 0 = x)$
- $\forall x \forall y (x + \sigma(y) = \sigma(x + y))$
- $\forall x (x \cdot 0 = 0)$
- $\forall x \forall y (x \cdot \sigma(y) = (x \cdot y) + x)$
- $\forall x \forall y (x < y \leftrightarrow \exists z (\sigma(z) + x = y))$

Induction axiom schema:

- $\forall y_1 \cdots \forall y_n (\varphi[0/x] \land \forall x (\varphi \rightarrow \varphi[\sigma(x)/x]) \rightarrow \forall x \varphi)$

— $\varphi$ is an arbitrary formula, where $\text{free}(\varphi) \subseteq \{x, y_1, \ldots, y_n\}$
Definitions

Let us assume a given set of axioms $\Gamma$.

- The definition of a predicate symbol $P$, where $\text{arity}(P) = n$:
  \[ \forall x_1 \cdots \forall x_n (P(x_1, \ldots, x_n) \leftrightarrow \varphi) \]
  where $\text{free}(\varphi) \subseteq \{x_1, \ldots, x_n\}$

- The definition of a function symbol $f$, where $\text{arity}(f) = n$:
  \[ \forall x_1 \cdots \forall x_n \forall y (f(x_1, \ldots, x_n) = y \leftrightarrow \varphi) \]
  where $\text{free}(\varphi) \subseteq \{x_1, \ldots, x_n, y\}$, and where $\Gamma \vdash \forall x_1 \cdots \forall x_n \exists y_1 y(\varphi)$

- The definition of a constant symbol $c$:
  \[ \varphi[c/x] \]
  where $\text{free}(\varphi) \subseteq \{x\}$, and where $\Gamma \vdash \exists x_1 x(\varphi)$
Signature $(\emptyset, \{\circ\}, \{e\}, \text{arity})$, where $\text{arity}(\circ) = 2$

Axioms:

- $\forall x \forall y \forall z ((x \circ y) \circ z = x \circ (y \circ z))$
- $\forall x (x \circ e = x)$
- $\forall x \exists y (x \circ y = e)$
Set Theory

Signature \((\in, \emptyset, \emptyset, \text{arity})\), where \(\text{arity}(\in) = 2\)

Examples of some of axioms of set theory:

- **Axiom of extensionality:**
  \[
  \forall x \forall y (\forall z (z \in x \iff z \in y) \rightarrow x = y)
  \]

- **Axiom schema of specification:**
  \[
  \forall y \exists z \forall x (x \in z \iff (x \in y \land \varphi))
  \]

  where \(\varphi\) is an arbitrary formula, where \(z \notin \text{free}(\varphi)\)

  The set of those elements \(x\) of set \(y\), for which it holds that \(\varphi\):
  \[
  \{ x \in y \mid \varphi \}
  \]
Often Used Abbreviations

- Proposition “\( \varphi \) holds for some element \( x \) of set \( A \)”:
  \[
  \exists x (x \in A \land \varphi)
  \]
  Abbreviation:
  \[(\exists x \in A)(\varphi)\]

- Proposition “\( \varphi \) holds for each element \( x \) of set \( A \)”:
  \[
  \forall x (x \in A \rightarrow \varphi)
  \]
  Abbreviation:
  \[(\forall x \in A)(\varphi)\]
Formal Languages
Alphabet and Word

**Definition**

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

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

**Definition**

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

**Example 1:**


Words over alphabet $\Sigma$: HELLO ABRACADABRA ERROR
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, 1010101010101010

Example 5:

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

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

Example of a word:

```java
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```
Language — a set of (some) words of symbols from a given alphabet.

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
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**.
Some Basic Concepts

The **length of a word** is the number of symbols of the word. For example, the length of word \(abaab\) is 5.

The length of a word \(w\) is denoted \(|w|\). For example, if \(w = abaab\) 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 **OST** and **RAVA** is the word **OSTRAVA**.

The operation of concatenation is denoted by symbol $\cdot$ (similarly to multiplication). It is possible to omit this symbol.

$$\text{OST} \cdot \text{RAVA} = \text{OSTRAVA}$$

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

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

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

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

\[ u \cdot v \neq v \cdot u \]

**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:

\[ \varepsilon \cdot w = w \cdot \varepsilon = w \]
Prefixes, Suffixes, and Subwords

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 \( y = u xv \).

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 \( \varepsilon, a, b, ab, ba, aba, baa, aab, abaa, baab, abaab \).
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.
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 \(\text{baa}\)
- \(L_2\) — 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 \(\text{baa}\) or an even number of occurrences of \(b\)
- \(L_1 \cap L_2\) — the set of all words containing subword \(\text{baa}\) or an even number of occurrences of \(b\)
- \(\overline{L_1}\) — the set of all words that do not contain subword \(\text{baa}\)
- \(L_1 - L_2\) — the set of all words that contain subword \(\text{baa}\) 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:

$abba$  $abbab$  $abbbbb$  $baa$  $baab$  $babbb$
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

$$\exists n \in \mathbb{N} : \exists w_1, w_2, \ldots, w_n \in L : w = w_1w_2\cdots w_n$$

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).
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

$$
\begin{align*}
L^0 &= \{\varepsilon\} \\
L^1 &= L \\
L^2 &= L \cdot L \\
L^3 &= L \cdot L \cdot L \\
L^4 &= L \cdot L \cdot L \cdot L \\
L^5 &= L \cdot L \cdot L \cdot L \cdot L \\
\ldots
\end{align*}
$$

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

$$aaaaaa \quad aaaaab \quad aabaa \quad aabb \quad baaaa \quad baab \quad bbbaa \quad bbb$$
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 $w = a_1 a_2 \cdots a_n$ (where $a_i \in \Sigma$) is $w^R = a_n a_{n-1} \cdots a_1$. 
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: $L = \{ab, baaba, aaab\}$

$L^R = \{ba, abaab, baaa\}$
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 $\Sigma^*$:

$x <_L y$ 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 $<_L$. 
Regular Expressions
Regular Expressions

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

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

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

- There are no other regular expressions except those defined in the two points mentioned above.
Regular Expressions

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

- According to the definition, 0 and 1 are regular expressions.
Example: $\Sigma = \{0, 1\}$

- 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: \[ \Sigma = \{0, 1\} \]

- 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.
Regular Expressions

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

- 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.
Regular Expressions

Example: \( \Sigma = \{0, 1\} \)

- 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 \( \alpha \).

\[\{(0 + 1) \cdot (0^*)\} = \{0, 1, 00, 10, 000, 100, 0000, 1000, 00000, \ldots\}\]
Regular Expressions

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)^*))\)
Regular Expressions

A description of an (abstract) syntax of regular expressions using Backus-Naur form:

\[\alpha ::= \emptyset \mid \varepsilon \mid a \mid \alpha^* \mid \alpha \cdot \alpha \mid \alpha + \alpha\]

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 superfluous due to associativity of operations of union (+) and concatenation (·).
- We can omit parentheses that are superfluous 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

\[
(01)^*111 + (00 + 1)^*
\]
Regular Expressions

**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 $\Sigma = \{0, 1\}$.

- $0 \ldots$ 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
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 $1$
- $0^*$ . . . the language containing words $\varepsilon$, $0$, $00$, $000$, . . .
Regular Expressions

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 $\varepsilon, 0, 00, 000, \ldots$
- $(01)^* \ldots$ the language containing words $\varepsilon, 01, 0101, 010101, \ldots$
Regular Expressions

**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, ...
- $(01)^* \ldots$ the language containing words $\varepsilon$, 01, 0101, 010101, ...
- $(0 + 1)^* \ldots$ 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 $\varepsilon, 0, 00, 000, \ldots$
- $(01)^* \ldots$ the language containing words $\varepsilon, 01, 0101, 010101, \ldots$
- $(0 + 1)^* \ldots$ the language containing all words over the alphabet $\{0, 1\}$
- $(0 + 1)^*00 \ldots$ the language containing all words ending with $00$
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 $1$
- $0^*$ ... the language containing words $\varepsilon, 0, 00, 000, \ldots$
- $(01)^*$ ... the language containing words $\varepsilon, 01, 0101, 010101, \ldots$
- $(0 + 1)^*$ ... the language containing all words over the alphabet $\{0, 1\}$
- $(0 + 1)^*00$ ... the language containing all words ending with $00$
- $(01)^*111(01)^*$ ... the language containing all words that contain a subword $111$ preceded and followed by an arbitrary number of copies of the word $01$
Regular Expressions

\[(0 + 1)^*00 + (01)^*111(01)^*\] ... 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 subword 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)* ... 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*(10*10*)* ... the language containing all words with an even number of occurrences of symbol 1
Finite Automata
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.
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.
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.
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.

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

\( \boxed{\text{YES}} \)
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.
Recognition of a Language

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 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 read so far is even or odd (i.e., it is sufficient to remember only the last bit of the number).

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

0
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 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:

\[ \text{E} \quad \quad \text{O} \]
Recognition of a Language

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

![Graph Diagram]

E  O

0
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:

![Diagram of a graph with nodes E and O, edges labeled 0 and 1]
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:
Recognition of a Language

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

![Graph](image-url)

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

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

![Diagram of a language recognition device]
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:

![Diagram of a language recognition device]

- E (initial state)
- O (final state)
- Transitions: 0→0, 1→1
- Input sequence: 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:
Recognition of a Language

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

![Graph Diagram]
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:

\[
\begin{array}{c}
\text{E} \\
0 \quad 1 \\
1 \\
\text{O} \\
0 \\
1 \\
\end{array}
\]

0 1 0 1 1 1 0 1 0 0 1

E
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:
Recognition of a Language

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

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

Z. Sawa (TU Ostrava)
Recognition of a Language

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.
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

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

$\delta(1, a) = 2 \quad \delta(1, b) = 1$
$\delta(2, a) = 4 \quad \delta(2, b) = 5$
$\delta(3, a) = 1 \quad \delta(3, b) = 4$
$\delta(4, a) = 1 \quad \delta(4, b) = 3$
$\delta(5, a) = 4 \quad \delta(5, b) = 5$
Deterministic Finite Automaton

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:

\[
\begin{array}{c|cc}
\delta & a & b \\
\hline
\leftrightarrow 1 & 2 & 1 \\
2 & 4 & 5 \\
3 & 1 & 4 \\
\leftrightarrow 4 & 1 & 3 \\
\leftrightarrow 5 & 4 & 5
\end{array}
\]
Deterministic Finite Automaton
Deterministic Finite Automaton

The diagram represents a Deterministic Finite Automaton (DFA). The states are labeled 1, 2, 3, 4, and 5. The transitions are as follows:

- From state 1, on input 'a', move to state 2.
- From state 1, on input 'b', loop back to state 1.
- From state 2, on input 'a', move to state 1.
- From state 2, on input 'b', move to state 3.
- From state 3, on input 'b', move to state 4.
- From state 4, on input 'a', move to state 3.
- From state 4, on input 'b', move to state 5.
- From state 5, on input 'b', loop back to state 5.

The initial state is 1, and the final state is 3.
Deterministic Finite Automaton

\[ a \xrightarrow{a} 2 \xrightarrow{b} 5 \]

\[ 1 \]

\[ a \quad b \quad a \quad b \quad b \]

\[ Z. \text{ Sawa (TU Ostrava)} \]

Introd. to Theoretical Computer Science

February 6, 2014 146 / 401
Deterministic Finite Automaton

\[
\begin{array}{c}
\text{1} \xrightarrow{a} \text{2} \xrightarrow{b} \text{5} \xrightarrow{a} \text{4}
\end{array}
\]
Deterministic Finite Automaton

[Diagram of a Deterministic Finite Automaton with labeled states and transitions: 1 → a 2, 2 → b 5, 3, 4 → a 5, 5 → b 3, transitions a → 1, b → 2, a → 5, b → 4, b → 3, 3]
Deterministic Finite Automaton

Z. Sawa (TU Ostrava)  Introd. to Theoretical Computer Science  February 6, 2014  146 / 401
### 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 $L(A) = L$. 
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.
Let us have the following two automata:

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

Do both of them accept the word \textit{ababb}?
An Automaton for Intersection of Languages

Let us have the following two automata:

\[
\begin{array}{c}
\begin{array}{c}
0 \quad b \\
1 \quad a \\
2 \quad b \\
3 \quad a, b
\end{array} \\
\textcolor{blue}{a} \\
\textcolor{blue}{b}
\end{array}
\quad \\
\begin{array}{c}
A \quad a, b \\
B \quad b
\end{array}
\]

Do both of them accept the word \textcolor{red}{ababb}?
An Automaton for Intersection of Languages

Let us have the following two automata:

Do both of them accept the word \textit{ababb}?
An Automaton for Intersection of Languages

Let us have the following two automata:

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

![Automata Diagram]

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

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

\[ \begin{align*}
0 & \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{a} 3 \\
& \xrightarrow{b} 0 \\
A & \xrightarrow{a} B \\
& \xrightarrow{b} b
\end{align*} \]
An Automaton for Intersection of Languages

\[ 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{a} 3 \]

\[ A \xrightarrow{a} B \]

\[ 0A \xrightarrow{a} 1B \]
An Automaton for Intersection of Languages

Z. Sawa (TU Ostrava)  Introd. to Theoretical Computer Science  February 6, 2014  156 / 401
An Automaton for Intersection of Languages

\begin{center}
\begin{tikzpicture}
    \node[state,initial] (0) at (0,0) {$0$};
    \node[state] (1) at (2,0) {$1$};
    \node[state] (2) at (4,0) {$2$};
    \node[state] (3) at (6,0) {$3$};
    \node[state,accepting] (A) at (8,0) {$A$};
    \node[state,accepting] (B) at (10,0) {$B$};
    \node[state,accepting] (0A) at (-2,-4) {$0A$};
    \node[state,accepting] (1B) at (2,-4) {$1B$};
    \node[state,accepting] (1A) at (4,-4) {$1A$};

    \draw[->] (0) edge node {$a$} (1);
    \draw[->] (1) edge node {$b$} (2);
    \draw[->] (2) edge node {$a$} (3);
    \draw[->] (3) edge node {$a, b$} (0);
    \draw[->] (A) edge node {$a$} (B);
    \draw[->] (0A) edge node {$a$} (1A);
    \draw[->] (0A) edge [bend right] node {$b$} (1B);
    \draw[->] (1B) edge node {$a$} (1A);
\end{tikzpicture}
\end{center}
An Automaton for Intersection of Languages

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

Z. Sawa (TU Ostrava)  Introd. to Theoretical Computer Science  February 6, 2014  156 / 401
An Automaton for Intersection of Languages
An Automaton for Intersection of Languages

Z. Sawa (TU Ostrava)  Introd. to Theoretical Computer Science  February 6, 2014  156 / 401
An Automaton for Intersection of Languages
An Automaton for Intersection of Languages

A transitions diagram is shown with states labeled 0, 1, 2, 3, A, and B. The edges indicate transitions on symbols a and b between these states. The automaton is designed to recognize the intersection of two languages represented by the separate automata on the right. The states and transitions are as follows:

- State 0 transitions on a to state 1 and on b to itself.
- State 1 transitions on a to state 2 and on b to state 1.
- State 2 transitions on a to state 3 and on b to state 2.
- State 3 transitions on a to state 2 and on b to state 3.
- State A transitions on a to state A and on b to state B.
- State B transitions on a to state B and on b to state B.

The automaton on the left intersects with the automata on the right at states 2 and 3, forming the intersection of the two languages.
An Automaton for Intersection of Languages
An Automaton for Intersection of Languages

Z. Sawa (TU Ostrava)
Introd. to Theoretical Computer Science
February 6, 2014 156 / 401
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

Z. Sawa (TU Ostrava)  Introd. to Theoretical Computer Science  February 6, 2014  156 / 401
An Automaton for Intersection of Languages

[Diagram of automata]
An Automaton for Intersection of Languages
An Automaton for 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.,

\[
L(A) = L(A_1) \cap L(A_2)
\]
Intersection of Regular Languages

**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
An Automaton for the Union of Languages

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

Z. Sawa (TU Ostrava)
The construction of an automaton $A$ that accepts the union of languages accepted by automata $A_1$ and $A_2$, i.e., the language

$$L(A_1) \cup L(A_2)$$

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

![Automaton Diagram]

- **States:** 0, 1, 2, 3
- **Symbols:** a, b, a, b
- **Transitions:**
  - 0: a → 1, b → 3
  - 1: a → 2
  - 2: a → 3
  - 3: a, b

Z. Sawa (TU Ostrava)
An Automaton for the Complement of a Language

Z. Sawa (TU Ostrava)

Introd. to Theoretical Computer Science

February 6, 2014 161 / 401
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 $w \notin L(A)$, 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 \( w \notin L(A) \), 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
Nondeterministic Finite Automaton
Nondeterministic Finite Automaton

\begin{center}
\begin{tikzpicture}[node distance=2cm,auto,initial text=]
  
  \node[state] (1) {1};
  \node[state] (2) [right of=1] {2};
  \node[state] (3) [below of=1] {3};
  \node[state] (4) [below of=2] {4};
  \node[state,accepting] (5) [right of=4] {5};

  \draw (1) edge [above] node {$a$} (2)
        edge [left] node {$a$} (3)
        edge [below] node {$b$} (4);
  \draw (2) edge [right] node {$b$} (5)
        edge [above] node {$a$} (4);
  \draw (3) edge [below] node {$b$} (4)
        edge [left] node {$a$} (1);
  \draw (4) edge [above] node {$a$} (2)
        edge [right] node {$b$} (5);
  \draw (5) edge [bend left] node {$b$} (5);
\end{tikzpicture}
\end{center}

\begin{center}
\begin{tabular}{cccccc}
  a & b & a & b & b \\
\end{tabular}
\end{center}

\begin{center}
\begin{align*}
  1 & \xrightarrow{a} 3 & \xrightarrow{b} 4
\end{align*}
\end{center}
Nondeterministic Finite Automaton

1 → a → 3 → b → 4 → a → 2

1 2 3 4 5
Nondeterministic Finite Automaton

1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{a} 2 \xrightarrow{b} 5
Nondeterministic Finite Automaton

\[ a \rightarrow 3 \quad b \rightarrow 4 \quad a \rightarrow 2 \quad b \rightarrow 5 \quad b \rightarrow 5 \]
Nondeterministic Finite Automaton

\begin{center}
\begin{tikzpicture}[node distance=2cm, >=latex]
    \node (1) [state, fill=blue!20] {1};
    \node (2) [state, fill=yellow!20, right of=1] {2};
    \node (3) [state, fill=yellow!20, below of=2] {3};
    \node (4) [state, fill=yellow!20, below of=2, right of=2] {4};
    \node (5) [state, fill=yellow!20, right of=4] {5};

    \path (1) edge [above] node {a} (2);
    \path (1) edge [below] node {b} (3);
    \path (2) edge [above] node {a} (4);
    \path (2) edge [right] node {b} (5);
    \path (3) edge [above] node {a} (4);
    \path (3) edge [below] node {b} (2);
    \path (4) edge [above] node {a} (5);
    \path (4) edge [right] node {b} (3);
    \path (5) edge [loop right] node {b} (5);
\end{tikzpicture}
\end{center}

\begin{center}
\begin{tabular}{cccccc}
    a & b & a & b & b & 1
\end{tabular}
\end{center}
Nondeterministic Finite Automaton

[Diagram of a nondeterministic finite automaton with states 1, 2, 3, 4, and 5. Edges labeled with 'a' and 'b' transition to different states.]

Z. Sawa (TU Ostrava) Introd. to Theoretical Computer Science February 6, 2014 164 / 401
Nondeterministic Finite Automaton

[Diagram of a nondeterministic finite automaton with states 1, 2, 3, 4, and 5, and transitions labeled with 'a' and 'b'.]

Z. Sawa (TU Ostrava) | Introd. to Theoretical Computer Science | February 6, 2014 | 164 / 401
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 \textit{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

\[
\begin{array}{c}
1 \\
2 & b & b \\
3 & a & a, b \end{array}
\]

\[
\begin{array}{c}
a, a, b, b, a \end{array}
\]

1, 2
Transformation of NFA to DFA
Transformation of NFA to DFA

\[ a \rightarrow 1, 2, 3 \]

1. Transition diagram:
   - From state 1:
     - On input 'a': Transition to state 2
     - On input 'b': Transition to state 3
   - From state 2:
     - On input 'a': Transition to state 1
     - On input 'b': Transition to state 3
   - From state 3:
     - On input 'a': Transition to state 2
     - On input 'b': Transition to state 1

2. Transition table:

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

Z. Sawa (TU Ostrava)
Introd. to Theoretical Computer Science
February 6, 2014 168 / 401
Transformation of NFA to DFA

Graphical representation of the transformation process.
Transformation of NFA to DFA

\[ \begin{array}{ccc}
\text{a} & \text{a} & \text{b} & \text{b} & \text{a} \\
\end{array} \]
Transformation of NFA to DFA
Transformation of NFA to DFA

\[
\begin{array}{c}
\{1,2\} \\
\rightarrow \\
1 \quad \leftrightarrow \quad 2 \quad \leftrightarrow \quad 3 \\
\quad \downarrow \quad a, b \\
\quad \uparrow \quad a, b \\
\quad \uparrow \quad a \\
\quad \downarrow \quad b \\
\quad \downarrow \quad b \\
\end{array}
\]
Transformation of NFA to DFA

\[ (1, 2) \xrightarrow{a} (2, 3) \]

\[ (2, 3) \xrightarrow{a, b} (3) \]

\[ (3) \xrightarrow{b} (1) \]

\[ (1) \xrightarrow{a} (2, 3) \]
Transformation of NFA to DFA
Transformation of NFA to DFA

Z. Sawa (TU Ostrava) Introd. to Theoretical Computer Science February 6, 2014 169 / 401
Transformation of NFA to DFA
Transformation of NFA to DFA
Transformation of NFA to DFA

Z. Sawa (TU Ostrava)
Introd. to Theoretical Computer Science
February 6, 2014 169 / 401
Transformation of NFA to DFA

\[
\begin{align*}
\text{1} & \xrightarrow{a} \text{2} & \text{1} & \xrightarrow{b} \text{3} \\
\text{2} & \xrightarrow{a} \text{3} \\
\text{3} & \\
\end{align*}
\]

\[
\begin{align*}
\text{\{1, 2\}} & \xrightarrow{a, b} \text{\{2, 3\}} & \text{\{2, 3\}} & \xrightarrow{b} \text{\{3\}} \\
\text{\{1, 2, 3\}} & \xrightarrow{a} \text{\{1, 2, 3\}} & \\
\text{\{1\}} & \xrightarrow{a} \text{\{1\}} \\
\end{align*}
\]
Transformation of NFA to DFA

Z. Sawa (TU Ostrava) Introd. to Theoretical Computer Science February 6, 2014
Transformation of NFA to DFA

Z. Sawa (TU Ostrava)  Introd. to Theoretical Computer Science  February 6, 2014  169 / 401
Transformation of NFA to DFA

\[
\begin{align*}
\{1, 2\} &\quad \text{a, b} \quad \{1, 2, 3\} \\
\{2, 3\} &\quad \text{b} \quad \{1\} \\
\{3\} &\quad \text{b} \quad \emptyset
\end{align*}
\]
### 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>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>
### 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>

$\leftrightarrow \{1, 2\}$
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>

\[
\leftrightarrow \{1, 2\} \quad \{2, 3\}
\]
### 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}, {2, 3}</td>
<td>{2, 3}</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>
<tr>
<td>← 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>← 2</td>
<td>{2,3}</td>
<td>{2,3}</td>
</tr>
<tr>
<td>← 3</td>
<td>{2,3}</td>
<td>{2,3}</td>
</tr>
</tbody>
</table>
Transformation of NFA to DFA

\[
\begin{array}{c|cc}
\leftrightarrow & a & b \\
1 & \rightarrow & 2, 3 \\
\rightarrow & 2 & 3 \\
3 & 1 & \rightarrow \\
\end{array}
\]

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

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

<table>
<thead>
<tr>
<th>State</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>State</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>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava)  
Introd. to Theoretical Computer Science  
February 6, 2014  
170 / 401
### 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>{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>↔ 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>

\[
\begin{array}{ccc}
\{1, 2\} & \{2, 3\} & \{2, 3\} \\
\{2, 3\} & \{1, 2, 3\} & \{3\} \\
\{1, 2, 3\} & \{1, 2, 3\} & \{2, 3\} \\
\{3\} & \{2, 3\} & \\
\end{array}
\]
### Transformation of NFA to DFA

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(-)</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>
</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>

<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>{3}</td>
</tr>
<tr>
<td>1</td>
<td>{1, 2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>2</td>
<td>{1}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>3</td>
<td>{3}</td>
<td>{1}</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>–</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</td>
<td>{2,3}</td>
<td>{2,3}</td>
</tr>
<tr>
<td>2</td>
<td>{1,2,3}</td>
<td>{3}</td>
</tr>
<tr>
<td>3</td>
<td>{1,2,3}</td>
<td>{2,3}</td>
</tr>
<tr>
<td>1</td>
<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>(\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>(\leftrightarrow {2, 3})</td>
<td>{1, 2, 3}</td>
<td>{3}</td>
</tr>
<tr>
<td>(\leftrightarrow {1, 2, 3})</td>
<td>{1, 2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>(\leftrightarrow {3})</td>
<td>{1}</td>
<td>{}</td>
</tr>
<tr>
<td>(\leftrightarrow {1})</td>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>(\leftrightarrow {})</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></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</td>
<td>{2, 3}</td>
<td>{2, 3}</td>
</tr>
<tr>
<td>2</td>
<td>{1, 2, 3}</td>
<td>{3}</td>
</tr>
<tr>
<td>3</td>
<td>{1}</td>
<td>{}</td>
</tr>
<tr>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>
## Transformation of NFA to DFA

### Transition Table for States

<table>
<thead>
<tr>
<th>State</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>

### Transition Table for Sets

<table>
<thead>
<tr>
<th>State</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>$\leftrightarrow 1$</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>$\leftrightarrow {1,2}$</th>
<th>${2,3}$</th>
<th>${2,3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${2,3}$</td>
<td>${2,3}$</td>
<td>${2,3}$</td>
<td>${3}$</td>
</tr>
<tr>
<td>$\leftarrow {1,2,3}$</td>
<td>${1,2,3}$</td>
<td>${2,3}$</td>
<td>${3}$</td>
</tr>
<tr>
<td>${3}$</td>
<td>${1}$</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>$\leftarrow {1}$</td>
<td>$\emptyset$</td>
<td>${2,3}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</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>

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

\[
\begin{array}{c}
1 \xrightarrow{a} 2 \\
2 \xrightarrow{b} 3 \\
3 \xrightarrow{b} 5 \\
5 \xrightarrow{b} 4 \\
1 \xrightarrow{b} 3 \\
3 \xrightarrow{b} 4 \\
4 \xrightarrow{a} 1 \\
4 \xrightarrow{b} 5 \\
1 \xrightarrow{a} 3 \\
3 \xrightarrow{b} 4 \\
\end{array}
\]
Generalized Nondeterministic Finite Automaton

A generalized nondeterministic finite automaton (GNFA) is a type of automaton that allows transitions labeled with the empty word (denoted by $\varepsilon$). This property makes it useful for certain types of automata and computations.

The diagram shows a GNFA with states labeled 1 through 5, with transitions labeled with inputs a or b, and one transition labeled with the empty word $\varepsilon$. The transitions are as follows:

- From state 1, there is a transition on input a to state 3, and on input $\varepsilon$ to state 1.
- From state 2, there is a transition on input a to state 1, and on input b to state 5.
- From state 3, there is a transition on input a to state 1, and on input b to state 4.
- From state 4, there is a transition on input b to state 1.
- From state 5, there is a transition on input b to state 5.

At the bottom of the diagram, there is a sequence showing possible paths through the automaton:

1 → 3 → 4 → $\varepsilon$ → 1
Generalized Nondeterministic Finite Automaton

```
1 → a → 3 → b → 4 → ε → 1 → a → 2
```

Z. Sawa (TU Ostrava)
Introd. to Theoretical Computer Science
February 6, 2014
Generalized Nondeterministic Finite Automaton

\[
\begin{array}{c}
1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 1 \xrightarrow{a} 2 \xrightarrow{b} 5
\end{array}
\]
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**
- **$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(Q)$** 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 $q \in Q$. 
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.
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 \( \hat{\delta} : \mathcal{P}(Q) \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q) \) 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 \( Cl_\varepsilon(K) \) be the all states reachable from the states from the set \( K \) by some arbitrary sequence of \( \varepsilon \)-transitions.

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

- \( K \subseteq Cl_\varepsilon(K) \)
- For each \( q \in Cl_\varepsilon(K) \) it holds that \( \delta(q, \varepsilon) \subseteq Cl_\varepsilon(K) \).

Remark: Let us note that \( Cl_\varepsilon(Cl_\varepsilon(K)) = Cl_\varepsilon(K) \) for arbitrary \( K \).

Let us also note that in the case of NFA (where \( \delta(q, \varepsilon) = \emptyset \) for each \( q \in Q \)) is \( Cl_\varepsilon(K) = K \).
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 $a \in \Sigma$:
  \[ \delta'(K, a) = \text{Cl}_\varepsilon(\hat{\delta}(\text{Cl}_\varepsilon(K), a)) \]
- $q_0' = \text{Cl}_\varepsilon(I)$
- $F' = \{ K \in Q' \mid \text{Cl}_\varepsilon(K) \cap F \neq \emptyset \}$

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

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

\[ A_1: \]

\[ A_2: \]

\[ A: \]

\[ L(A) = L(A_1) \cdot L(A_2) \]
Concatenation of Languages

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

\[ A_1: \]

\[ A_2: \]

An incorrect construction:

\[ A: \]

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

$A_1$ and $A_2$
Concatenation of Languages

\[ A_1 \] \[
\begin{array}{c}
\text{\includegraphics[width=0.3\textwidth]{a1.png}} \end{array}
\]

\[ A_2 \] \[
\begin{array}{c}
\text{\includegraphics[width=0.3\textwidth]{a2.png}} \end{array}
\]

\[ A \] \[
\begin{array}{c}
\text{\includegraphics[width=0.6\textwidth]{a.png}} \end{array}
\]
Iteration of a Language

\[ A_1 \]

\[ A \]

\[ A_1 \]
An alternative construction for the union of languages:
An alternative construction for the union of languages:

$A_1$

$A_2$

$A$

$A_1$

$A_2$
The set of (all) regular languages is closed with respect to:

- union
- intersection
- complement
- concatenation
- iteration
- ...
Transformation of a Regular Expression to a Finite Automaton

**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:

∅
ε
a

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

The automata for the elementary expressions:

∅

ε

The construction for the union:
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:
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 $((0 + 1) \cdot 1)^*$:
Transformation of a Regular Expression to a Finite Automaton

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

**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.
Transformation of an Automaton to a Regular Expression

**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.
Transformation of an Automaton to a 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:

\[ \text{Transformation of an Automaton to a Regular Expression} \]
Transformation of an Automaton to a Regular Expression

Example:
Example:
Example:

\[ s \xrightarrow{a(b + aa)^*} 3 \xrightarrow{\varepsilon + (a + ba)(b + aa)^*} f \]

\[ b + a(b + aa)^*ab \]

\[ bb + (a + ba)(b + aa)^*ab \]
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.
Context-Free Grammars
Context-Free Grammars

Example:

\[
\begin{align*}
\langle \text{STMT} \rangle & \rightarrow \langle \text{IF-STM} \rangle \mid \langle \text{WHILE-STM} \rangle \mid \langle \text{BLOCK-STM} \rangle \mid \langle \text{ASSG-STM} \rangle \\
\langle \text{IF-STM} \rangle & \rightarrow \text{if} \ \langle \text{BOOL-EXP} \rangle \ \text{then} \ \langle \text{STMT} \rangle \ \text{else} \ \langle \text{STMT} \rangle \\
\langle \text{WHILE-STM} \rangle & \rightarrow \text{while} \ \langle \text{BOOL-EXP} \rangle \ \text{do} \ \langle \text{STMT} \rangle \\
\langle \text{BLOCK-STM} \rangle & \rightarrow \text{begin} \ \langle \text{STM-List} \rangle \ \text{end} \\
\langle \text{STM-List} \rangle & \rightarrow \langle \text{STMT} \rangle \mid \langle \text{STMT} \rangle ; \ \langle \text{STM-List} \rangle \\
\langle \text{ASSG-STM} \rangle & \rightarrow \langle \text{VAR} \rangle := \langle \text{ARITH-EXP} \rangle \\
\langle \text{BOOL-EXP} \rangle & \rightarrow \langle \text{ARITH-EXP} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXP} \rangle \\
\langle \text{COMPARE-OP} \rangle & \rightarrow \langle \text{LT} \rangle \mid \langle \text{GT} \rangle \mid \langle \text{LE} \rangle \mid \langle \text{GE} \rangle \mid \langle \text{EQ} \rangle \mid \langle \text{NE} \rangle \\
\langle \text{ARITH-EXP} \rangle & \rightarrow \langle \text{VAR} \rangle \mid \langle \text{CONST} \rangle \mid \left( \langle \text{ARITH-EXP} \rangle \langle \text{ARITH-OP} \rangle \langle \text{ARITH-EXP} \rangle \right) \\
\langle \text{ARITH-OP} \rangle & \rightarrow \langle \text{PLUS} \rangle \mid \langle \text{MINUS} \rangle \mid \langle \text{MUL} \rangle \mid \langle \text{DIV} \rangle \\
\langle \text{CONST} \rangle & \rightarrow \langle \text{NUM} \rangle \\
\langle \text{VAR} \rangle & \rightarrow \langle \text{LETTER} \rangle \\
\end{align*}
\]
The symbols, that have the form $\langle xxx \rangle$ in the previous example, are called **nonterminal symbols (nonterminals)**.

The rules describe the strings represented by a given nonterminal.

From nonterminal $\langle \text{STMT} \rangle$ we can for example obtain the text

```
while $x \leq y$ do begin $x := (x + 1)$; $y := (y - 1)$ end
```
while $x \leq y$ do begin $x := (x + 1)$; $y := (y - 1)$ end
while \( x \leq y \) do begin \( x := (x + 1) \); \( y := (y - 1) \) end

\langle STMT \rangle
\[ \textbf{while } x \leq y \textbf{ do begin } x := (x + 1); \ y := (y - 1) \textbf{ end} \]

\[ \langle \text{STMT} \rangle \]
\[ \langle \text{WHILE-STMT} \rangle \]
\textbf{Context-Free Grammars}

\begin{verbatim}
while \( x \leq y \) do begin \( x := (x + 1) \); \( y := (y - 1) \) end
\end{verbatim}

\[
\langle \text{STMT} \rangle \\
\langle \text{WHILE-STMT} \rangle \\
\textbf{while} \ \langle \text{BOOL-EXPR} \rangle \ \textbf{do} \ \langle \text{STMT} \rangle
\]
while $x \leq y$ do begin $x := (x + 1)$; $y := (y - 1)$ end

\langle \text{STMT} \rangle
\langle \text{WHILE-STMT} \rangle
\textbf{while} \langle \text{BOOL-EXPR} \rangle \textbf{ do } \langle \text{STMT} \rangle
\textbf{while} \langle \text{ARITH-EXPR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle \textbf{ do } \langle \text{STMT} \rangle
while $x \leq y$ do begin $x := (x + 1)$; $y := (y - 1)$ end

\[\langle \text{STMT} \rangle\]
\[\langle \text{WHILE-STMT} \rangle\]
\[\textbf{while} \ \langle \text{BOOL-EXPR} \rangle \ \textbf{do} \ \langle \text{STMT} \rangle\]
\[\textbf{while} \ \langle \text{ARITH-EXPR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle \ \textbf{do} \ \langle \text{STMT} \rangle\]
\[\textbf{while} \ \langle \text{VAR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle \ \textbf{do} \ \langle \text{STMT} \rangle\]
\[ \text{while } x \leq y \text{ do begin } x := (x + 1); y := (y - 1) \text{ end} \]

\[
\langle \text{STMT} \rangle
\]

\[
\langle \text{WHILE-STMT} \rangle
\]

\[
\text{while } \langle \text{BOOL-EXPR} \rangle \text{ do } \langle \text{STMT} \rangle
\]

\[
\text{while } \langle \text{ARITH-EXPR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle \text{ do } \langle \text{STMT} \rangle
\]

\[
\text{while } \langle \text{VAR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle \text{ do } \langle \text{STMT} \rangle
\]

\[
\text{while } \langle \text{VAR} \rangle \leq \langle \text{ARITH-EXPR} \rangle \text{ do } \langle \text{STMT} \rangle
\]
while $x \leq y$ do begin $x := (x + 1)$; $y := (y - 1)$ end

⟨STM⟩
⟨WHILE-STM⟩
while ⟨BOOL-EXPR⟩ do ⟨STM⟩
while ⟨ARITH-EXPR⟩⟨COMPARE-OP⟩⟨ARITH-EXPR⟩ do ⟨STM⟩
while ⟨VAR⟩⟨COMPARE-OP⟩⟨ARITH-EXPR⟩ do ⟨STM⟩
while ⟨VAR⟩ ≤ ⟨ARITH-EXPR⟩ do ⟨STM⟩
while ⟨VAR⟩ ≤ ⟨VAR⟩ do ⟨STM⟩
while $x \leq y$ do begin $x := (x + 1); y := (y - 1)$ end

⟨STMT⟩
⟨WHILE-STMT⟩
while ⟨BOOL-EXPR⟩ do ⟨STMT⟩
while ⟨ARITH-EXPR⟩⟨COMPARE-OP⟩⟨ARITH-EXPR⟩ do ⟨STMT⟩
while ⟨VAR⟩⟨COMPARE-OP⟩⟨ARITH-EXPR⟩ do ⟨STMT⟩
while ⟨VAR⟩ ≤ ⟨ARITH-EXPR⟩ do ⟨STMT⟩
while ⟨VAR⟩ ≤ ⟨VAR⟩ do ⟨STMT⟩
while $x \leq ⟨VAR⟩$ do ⟨STMT⟩
while \( x \leq y \) do begin \( x := (x + 1) \); \( y := (y - 1) \) end

\[
\langle \text{STMT} \rangle \\
\langle \text{WHILE-STMT} \rangle \\
\text{while} \ \langle \text{BOOL-EXPR} \rangle \ \text{do} \ \langle \text{STMT} \rangle \\
\text{while} \ \langle \text{ARITH-EXPR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle \ \text{do} \ \langle \text{STMT} \rangle \\
\text{while} \ \langle \text{VAR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle \ \text{do} \ \langle \text{STMT} \rangle \\
\text{while} \ \langle \text{VAR} \rangle \leq \langle \text{ARITH-EXPR} \rangle \ \text{do} \ \langle \text{STMT} \rangle \\
\text{while} \ \langle \text{VAR} \rangle \leq \langle \text{VAR} \rangle \ \text{do} \ \langle \text{STMT} \rangle \\
\text{while} \ \langle \text{VAR} \rangle \leq \langle \text{VAR} \rangle \ \text{do} \ \langle \text{STMT} \rangle \\
\text{while} \ x \leq \langle \text{VAR} \rangle \ \text{do} \ \langle \text{STMT} \rangle \\
\text{while} \ x \leq y \ \text{do} \ \langle \text{STMT} \rangle 
\]
while $x \leq y$ do begin $x := (x + 1)$; $y := (y - 1)$ end

⟨STMT⟩
⟨WHILE-STMT⟩
while ⟨BOOL-EXPR⟩ do ⟨STMT⟩
while ⟨ARITH-EXPR⟩⟨COMPARE-OP⟩⟨ARITH-EXPR⟩ do ⟨STMT⟩
while ⟨VAR⟩⟨COMPARE-OP⟩⟨ARITH-EXPR⟩ do ⟨STMT⟩
while ⟨VAR⟩ $\leq$ ⟨ARITH-EXPR⟩ do ⟨STMT⟩
while ⟨VAR⟩ $\leq$ ⟨VAR⟩ do ⟨STMT⟩
while $x \leq$ ⟨VAR⟩ do ⟨STMT⟩
while $x \leq y$ do ⟨STMT⟩
while $x \leq y$ do ⟨BLOCK-STMT⟩
while $x \leq y$ do begin $x := (x + 1)$; $y := (y - 1)$ end

\[ \langle \text{STMT} \rangle \]
\[ \langle \text{WHILE-STMT} \rangle \]
\[ \textbf{while} \ \langle \text{BOOL-EXPR} \rangle \ \textbf{do} \ \langle \text{STMT} \rangle \]
\[ \textbf{while} \ \langle \text{ARITH-EXPR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle \ \textbf{do} \ \langle \text{STMT} \rangle \]
\[ \textbf{while} \ \langle \text{VAR} \rangle \langle \text{COMPARE-OP} \rangle \langle \text{ARITH-EXPR} \rangle \ \textbf{do} \ \langle \text{STMT} \rangle \]
\[ \textbf{while} \ \langle \text{VAR} \rangle \ \leq \ \langle \text{ARITH-EXPR} \rangle \ \textbf{do} \ \langle \text{STMT} \rangle \]
\[ \textbf{while} \ \langle \text{VAR} \rangle \ \leq \ \langle \text{VAR} \rangle \ \textbf{do} \ \langle \text{STMT} \rangle \]
\[ \textbf{while} \ x \ \leq \ \langle \text{VAR} \rangle \ \textbf{do} \ \langle \text{STMT} \rangle \]
\[ \textbf{while} \ x \ \leq \ y \ \textbf{do} \ \langle \text{STMT} \rangle \]
\[ \textbf{while} \ x \ \leq \ y \ \textbf{do} \ \langle \text{BLOCK-STMT} \rangle \]
\[ \textbf{while} \ x \ \leq \ y \ \textbf{do} \ \textbf{begin} \ \langle \text{STMT-LIST} \rangle \ \textbf{end} \]
while $x \leq y$ do begin $x := (x + 1); y := (y - 1)$ end

⟨STMT⟩
⟨WHILE-STMT⟩
while ⟨BOOL-EXPR⟩ do ⟨STMT⟩
while ⟨ARITH-EXPR⟩ ⟨COMPARE-OP⟩ ⟨ARITH-EXPR⟩ do ⟨STMT⟩
while ⟨VAR⟩ ⟨COMPARE-OP⟩ ⟨ARITH-EXPR⟩ do ⟨STMT⟩
while ⟨VAR⟩ ≤ ⟨ARITH-EXPR⟩ do ⟨STMT⟩
while ⟨VAR⟩ ≤ ⟨VAR⟩ do ⟨STMT⟩
while $x \leq ⟨VAR⟩$ do ⟨STMT⟩
while $x \leq y$ do ⟨STMT⟩
while $x \leq y$ do ⟨BLOCK-STMT⟩
while $x \leq y$ do begin ⟨STMT-LIST⟩ end

...
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

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

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

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

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

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

\( A \)
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:

$A \Rightarrow aBBb$
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:

\[
A \Rightarrow aBBb \Rightarrow abCABb
\]
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:

$A \Rightarrow aBBb \Rightarrow abCABb$
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:

\( A \Rightarrow aBBb \Rightarrow abCA \overline{AB}b \)
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 \textit{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

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

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

\[ A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \]
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:

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBb
\]
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:
\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBbBb \Rightarrow abCaBbBb
\]
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 *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

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

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

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abbCABbBb
\]
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:

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abC_aBbBb \Rightarrow abbaBbBb
\]
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:

\[
\begin{align*}
A & \Rightarrow aBBb \\
& \Rightarrow abCABb \\
& \Rightarrow abCaBBbBb \\
& \Rightarrow abCaBbBb \\
& \Rightarrow abbaBbBb
\end{align*}
\]
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:

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb
\]
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:

\[
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb
\]
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 $ababbb$ 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

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

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

\[
\begin{align*}
A & \Rightarrow aBBb \\
& \Rightarrow abCABb \\
& \Rightarrow abCaBBbBb \\
& \Rightarrow abCaBbBb \\
& \Rightarrow abbaBbBb \\
& \Rightarrow abbaBbb \\
& \Rightarrow abbabb
\end{align*}
\]
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:

\[
\begin{align*}
A & \Rightarrow aBBb \\
& \Rightarrow abCABb \\
& \Rightarrow abCaBBbBb \\
& \Rightarrow abCaBbBb \\
& \Rightarrow abbaBbBb \\
& \Rightarrow abbaBbb \\
& \Rightarrow abbabb
\end{align*}
\]
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

\[ aCBbA \Rightarrow aCbCAbA \]

**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

\[
aC BbA \Rightarrow aC bCA bA
\]

**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

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

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

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

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

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 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:

\[
S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon
\]
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:

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

\[
S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow abaaaba
\]
**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 $)()()$ $\notin L$. 
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 $())() \notin L$.

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 $())()$ $\not\in L$.

Solution:

$$S \rightarrow \varepsilon \mid (S) \mid SS$$

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow (SS)(S) \Rightarrow ((S)S)(S) \Rightarrow ((()S)(S) \Rightarrow (((S)))(S) \Rightarrow (((())(S)) \Rightarrow (((())S)) \Rightarrow (((())())(S)) \Rightarrow (((())())(S)) \Rightarrow (((())())())$
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 $\ast$ can be used as operators.

For example $(a + a) \ast a + (a \ast a) \in L$. 
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 ‘\( \ast \)’ can be used as operators.

For example \((a + a) \ast a + (a \ast a) \in L\).

Solution:

\[
E \rightarrow a \mid E + E \mid E \ast E \mid (E)
\]
Context-Free Grammars

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:

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

$$E \Rightarrow E + E \Rightarrow E * E + E \Rightarrow (E) * E + E \Rightarrow (E + E) * E + E \Rightarrow (a + E) * E + E \Rightarrow (a + a) * E + E \Rightarrow (a + a) * a + E \Rightarrow (a + a) * a + (E) \Rightarrow (a + a) * a + (E * E) \Rightarrow (a + a) * a + (a * E) \Rightarrow (a + a) * a + (a * a)$$
Derivation Tree

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

A → aBBb | AaA
B → ε | bCA
C → AB | a | b
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 \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 aBBb
$A \rightarrow aBb$ \hspace{5mm} $B \rightarrow \epsilon \hspace{5mm} bCA$

$C \rightarrow AB \hspace{5mm} a \hspace{5mm} b$

$
A \Rightarrow aBBb \Rightarrow a\underbrace{bC}_A Bb
$

Derivation Tree
Derivation Tree

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

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

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

\[
A \Rightarrow aBBb \Rightarrow abCABb
\]
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

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

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

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

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

A ⇒ aBBb ⇒ abCABb ⇒ abCaBbBb ⇒ abCaBbBb
Derivation Tree

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

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

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

\[
\begin{align*}
A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abC_\text{a}BbBb
\end{align*}
\]
A → aBBb | AaA
B → ε | bCA
C → AB | a | b

A ⇒ aBBb ⇒ abCABb ⇒ abCaBBbBb ⇒ ab\textcolor{red}{C}aBbBb ⇒ ab\textcolor{red}{b}aBbBb
A → aBBb | AaA
B → ε | bCA
C → AB | a | b

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

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

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

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

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

A ⇒ aBBb ⇒ abCABb ⇒ abcBBbBb ⇒ abCaBbBb ⇒ abbaBbBb ⇒ abbaBbb
A \rightarrow aBBb \mid AaA
B \rightarrow \varepsilon \mid bCA
C \rightarrow AB \mid a \mid b

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

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

A ⇒ aBBb ⇒ abCABb ⇒ abCaBBbBb ⇒ abCaBbBb ⇒ abbaBbBb ⇒ abbaBbb ⇒ abbbabb
For each derivation there is some derivation 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 $A \rightarrow \alpha$. 
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:**

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

\[
E \Rightarrow E \ast E \Rightarrow E \ast E + E \Rightarrow a \ast E + E \Rightarrow a \ast a + E \Rightarrow a \ast a + a
\]
Sometimes it is possible to replace an ambiguous grammar with a grammar generating the same language but which is not ambiguous.

**Example:** A grammar

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

can be replaced with the equivalent grammar

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

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

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
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 $L(G) = L(G_1)^*$:

  $$G = (\Pi_1 \cup \{S\}, \Sigma, S, P_1 \cup \{S \rightarrow \varepsilon, S \rightarrow S_1S\})$$
Computability and Complexity
What Is an Algorithm?

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 \((In, Out, R)\), where:

- **In** is the set of possible inputs
- **Out** is the set of possible outputs
- \(R \subseteq In \times Out\) is a relation assigning corresponding outputs to each input. This relation must satisfy

\[
\forall x \in In : \exists y \in Out : R(x, y).
\]
In general, we can restrict to the case, where inputs and outputs of a problem are words over some $\Sigma$, i.e., $\text{In} = \text{Out} = \Sigma^*$. Some other object (numbers, sequences of numbers, graphs, . . . ,) 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 Diagram]

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’.
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 $\{\text{YES, NO}\}$ 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?
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 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 $\text{YES}$.

**Example:** A language consisting of those words from $\{0,1\}^*$ that are binary representations of primes.

For example, $101 \in L$ but $110 \notin L$. 
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 $\nu(\varphi_2) = 0$. 
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.
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 $(In, Out, f, m, g)$, where:

- $In$ is the set of inputs,
- $Out$ is the set of solutions,
- $f : In \rightarrow \mathcal{P}(Out)$ is a function assigning to each input $x$ a set of corresponding feasible solutions $f(x)$,
- $m : \bigcup_{x \in In} \left( \{x\} \times f(x) \right) \rightarrow \mathbb{R}$ is an optimization function (cost function),
- $g$ is min or max.

The goal is to find for a given input $x \in In$ 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., $f(x) = \emptyset$).
Optimization Problems

- The optimization problems, where \( g \) is \( \text{min} \), are called **minimization problems**.
- The optimization problems, where \( g \) is \( \text{max} \), are called **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 $G$?

![Graph Diagram](image)

$k = 4$

**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

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**: $k = 5$
ILP – Integer Linear Programming

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:

$$
\begin{align*}
3x_1 & - 2x_2 + 5x_3 \leq 8 \\
x_1 + x_3 & \leq -3 \\
2x_1 + x_2 & \leq 5
\end{align*}
$$
ILP – Integer Linear Programming

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 \textbf{YES}. 
### Examples of Problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Input: Deterministic finite automata $A_1$ and $A_2$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question: Is $L(A_1) = L(A_2)$?</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
<th>Input: Context-free grammars $G_1$ and $G_2$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question: Is $L(G_1) = L(G_2)$?</td>
<td></td>
</tr>
</tbody>
</table>
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 $P = (\text{In}, \text{Out}, R)$ and an algorithm $A$ realizing a function $f_A : \text{In} \rightarrow \text{Out}$, we say that

*the algorithm $A$ solves the problem $P$*

if:

- the value of $f_A(x)$ is defined for each $x \in \text{In}$,
- for each $x \in \text{In}$ 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 **algorithmically unsolvable**.

A decision problem that is not decidable is **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, ...; 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

output

working memory

0 0
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8

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:

\[ \text{LOAD } \langle \text{op} \rangle \]

reads the value of the operand \( \langle \text{op} \rangle \) into the working register (i.e., into cell 0).

\[ \text{LOAD } =5 \quad \text{– loads 5 into the working register} \]
\[ \text{LOAD } 5 \quad \text{– loads the value in cell 5 into the working register} \]
\[ \text{LOAD } \ast 5 \quad \text{– loads the value in cell } 5 + j, \text{ where } j \text{ is the current value of the index register, into the working register} \]
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)
Random Access Machine

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
output: WRITE
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
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:  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
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: 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

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: 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

\[
\begin{align*}
\text{start:} & \quad \text{READ} \\
& \quad \text{STORE} \quad 3 \\
& \quad \text{LOAD} \quad =1 \\
\text{loop:} & \quad \text{STORE} \quad 2 \\
& \quad \text{READ} \\
& \quad \text{JZERO} \quad \text{output} \\
& \quad \text{SUB} \quad 3 \\
& \quad \text{JZERO} \quad \text{found} \\
& \quad \text{LOAD} \quad 2 \\
& \quad \text{ADD} \quad =1 \\
& \quad \text{JUMP} \quad \text{loop} \\
\text{found:} & \quad \text{LOAD} \quad 2 \\
\text{output:} & \quad \text{WRITE} \\
& \quad \text{HALT}
\end{align*}
\]

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: -4
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0

Output:

Instructions: 16
Random Access Machine

\begin{align*}
\text{start:} & \quad \text{READ} \\
& \quad \text{STORE} \ 3 \\
& \quad \text{LOAD} \ =1 \\
\text{loop:} & \quad \text{STORE} \ 2 \\
& \quad \text{READ} \\
& \quad \text{JZERO} \ \text{output} \\
& \quad \text{SUB} \ 3 \\
& \quad \text{JZERO} \ \text{found} \\
& \quad \text{LOAD} \ 2 \\
& \quad \text{ADD} \ =1 \\
& \quad \text{JUMP} \ \text{loop} \\
\text{found:} & \quad \text{LOAD} \ 2 \\
\text{output:} & \quad \text{WRITE} \\
& \quad \text{HALT} \\
\end{align*}

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
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: 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
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.
Formally, a **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)
- $\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$ is a **transition function**
- $q_0 \in Q$ is an **initial state**
- $F \subseteq Q$ is a set of **final states**
Turing Machine

- 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
Turing Machine

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 $L(\mathcal{M})$ of a Turing machine $\mathcal{M}$ is the set of all words accepted by $\mathcal{M}$. 
A Turing machine can give not only an answer \textsc{Yes} or \textsc{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 \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language $L = \{ a^n b^n c^n \mid n \geq 0 \}$

$Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \}$

$F = \{ q_{\text{yes}}, q_{\text{no}} \}$

$\Sigma = \{ a, b, c \}$

$\Gamma = \{ \square, a, b, c, x \}$

<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>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_3, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</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>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>

$q_1$
Turing Machine

Language \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\( \Sigma = \{ a, b, c \} \quad \Gamma = \{ \square, a, b, c, x \} \)

\[ \begin{array}{c|cccc|cccc|cccc|cccc} \delta & \square & a & b & c & x \\ \hline q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\ q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\ q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\ q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\ q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \end{array} \]
**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_{\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>
<tr>
<td>( q_1 )</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, +1 )</td>
</tr>
<tr>
<td>( q_2 )</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, +1 )</td>
</tr>
<tr>
<td>( q_3 )</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, +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, +1 )</td>
</tr>
</tbody>
</table>

\( q_1 \)
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\sqcup, a, b, c, x\} \)

\[
\begin{array}{|c|cccc|}
\hline
\delta & \sqcup & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \)
\( F = \{q_{\text{yes}}, q_{\text{no}}\} \)
\( \Sigma = \{a, b, c\} \quad \Gamma = \{\Box, a, b, c, x\} \)

\[
\begin{array}{|c|cccccc|}
\hline
\delta & \Box & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \Box, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \Box, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \Box, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \Box, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \Box, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]
**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

\[
\begin{array}{|c|cccc|}
\hline
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_{\text{no}}, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_{\text{no}}, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]

- \( \square \) - Initial state
- \( x \) - Blank symbol
- \( a \) - Input symbols
- \( b \) - Input symbols
- \( c \) - Input symbols
- \( \square \) - Blank symbol
## Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

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

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \]

\[ F = \{q_{\text{yes}}, q_{\text{no}}\} \]

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

\[ \Gamma = \{\square, a, b, c, x\} \]

<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_{\text{no}}, b, 0 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>(( q_{\text{no}}, \square, 0 ))</td>
<td>(( q_{\text{no}}, a, 0 ))</td>
<td>(( q_{\text{no}}, a, 0 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</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_{\text{no}}, a, 0 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</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_{\text{no}}, c, +1 ))</td>
<td>(( q_{\text{no}}, c, +1 ))</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>(( q_{\text{no}}, \square, +1 ))</td>
<td>(( q_{\text{no}}, a, -1 ))</td>
<td>(( q_{\text{no}}, a, -1 ))</td>
<td>(( q_{\text{no}}, c, -1 ))</td>
<td>(( q_{\text{no}}, c, -1 ))</td>
</tr>
</tbody>
</table>

\[ Z. Sawa (TU Ostrava) \]

Introd. to Theoretical Computer Science

February 6, 2014 264 / 401
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_0, 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_{\text{no}}, 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, -1 ))</td>
<td>(( q_4, c, -1 ))</td>
<td>(( q_4, x, -1 ))</td>
</tr>
</tbody>
</table>

- Initial state: \( q_0 \)
- Final states: \( F = \{q_{\text{yes}}, q_{\text{no}}\} \)
- Input alphabet: \( \Sigma = \{a, b, c\} \)
- Tape alphabet: \( \Gamma = \{\square, a, b, c, x\} \)

Diagram:

- Initial tape: \( \square \square x a a a x b b b b c c c c c \)
- Current state: \( q_2 \)

Z. Sawa (TU Ostrava)  Introd. to Theoretical Computer Science  February 6, 2014  264 / 401
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 $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  \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 \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{|c|c|c|c|c|}
\hline
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]
**Turing Machine**

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<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>

**Diagram:**

[Diagram showing the transition from state \( q_3 \) to \( q_3 \) with input symbols.]
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>$(q_{\text{yes}}, #, 0)$</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>$(q_{\text{no}}, #, -1)$</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>
**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 \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
## Language

Language $L = \{ a^n b^n c^n \mid n \geq 0 \}$

- $Q = \{ q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no} \}$
- $F = \{ q_{yes}, q_{no} \}$
- $\Sigma = \{ a, b, c \}$
- $\Gamma = \{ \Box, a, b, c, x \}$

<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>$(q_2, b, +1)$</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>$(q_{no}, a, 0)$</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>$(q_0, \Box, +1)$</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>

### Example

- Initial state: $q_4$
- Input: $\Box \ Box \ x \ a \ a \ a \ x \ b \ b \ b \ x \ c \ c \ c \ \Box \ \Box \ \Box$
- Transition: $(q_0, \Box, +1)$
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 $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\}$ \quad $F = \{q_{yes}, q_{no}\}$

$\Sigma = \{a, b, c\}$ \quad $\Gamma = \{\blacksquare, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\blacksquare$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{yes}, \blacksquare, 0)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_{no}, a, 0)$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{no}, \blacksquare, 0)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_{no}, a, 0)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{no}, \blacksquare, 0)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_{no}, a, 0)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_{no}, \blacksquare, -1)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, +1)$</td>
<td>$(q_{no}, a, 0)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_{no}, \blacksquare, +1)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, +1)$</td>
<td>$(q_{no}, a, 0)$</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_{\text{no}}, b, 0) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_0, x, +1) )</td>
<td></td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_1, x, +1) )</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_{\text{no}}, b, +1) )</td>
<td>( (q_3, x, +1) )</td>
<td></td>
</tr>
<tr>
<td>( q_3 )</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>
</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, +1) )</td>
<td>( (q_4, x, +1) )</td>
<td></td>
</tr>
</tbody>
</table>

\[ Z. Sawa (TU Ostrava) \quad \text{Introd. to Theoretical Computer Science} \quad \text{February 6, 2014} \quad 264 / 401 \]
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_{\text{no}}, b, 0 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>

*Z. Sawa (TU Ostrava)  Introd. to Theoretical Computer Science  February 6, 2014  264 / 401*
### 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 $\Gamma = \{\square, a, b, c, x\}$

<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 a a a x b b b b x c c c □ □ □ □
```

$q_4$
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{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>($q_4, a, -1$)</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 $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\}$ \hspace{1cm} $F = \{q_{yes}, q_{no}\}$

$\Sigma = \{a, b, c\}$ \hspace{1cm} $\Gamma = \{\square, a, b, c, x\}$

$$
\begin{array}{c|ccccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{yes}, \square, 0) & (q_1, x, +1) & (q_{no}, b, 0) & (q_{no}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{no}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{no}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{no}, \square, 0) & (q_{no}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_{no}, \square, -1) & (q_{no}, a, 0) & (q_{no}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_{no}, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
$$
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>( \text{a} )</th>
<th>( \text{b} )</th>
<th>( \text{c} )</th>
<th>( \text{x} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( (q_{\text{yes}}, \square, 0) )</td>
<td>( (q_1, \text{x}, +1) )</td>
<td>( (q_{\text{no}}, \text{b}, 0) )</td>
<td>( (q_{\text{no}}, \text{c}, 0) )</td>
<td>( (q_0, \text{x}, +1) )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_1, \text{a}, +1) )</td>
<td>( (q_2, \text{x}, +1) )</td>
<td>( (q_{\text{no}}, \text{c}, 0) )</td>
<td>( (q_1, \text{x}, +1) )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_{\text{no}}, \text{a}, 0) )</td>
<td>( (q_2, \text{b}, +1) )</td>
<td>( (q_3, \text{x}, +1) )</td>
<td>( (q_2, \text{x}, +1) )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( (q_{\text{yes}}, \square, -1) )</td>
<td>( (q_{\text{no}}, \text{a}, 0) )</td>
<td>( (q_{\text{no}}, \text{b}, 0) )</td>
<td>( (q_3, \text{c}, +1) )</td>
<td>( (q_3, \text{x}, +1) )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( (q_{\text{yes}}, \square, +1) )</td>
<td>( (q_4, \text{a}, -1) )</td>
<td>( (q_4, \text{b}, -1) )</td>
<td>( (q_4, \text{c}, -1) )</td>
<td>( (q_4, \text{x}, -1) )</td>
</tr>
</tbody>
</table>

\[ q_4 \]
**Turing Machine**

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad \text{and} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<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:**

```
□ □ x a a a x b b b x c c c □ □ □
```

```
q0
```
**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_{\text{no}}, b, 0 ))</td>
<td>(( q_{\text{no}}, c, 0 ))</td>
<td>(( q_{\text{no}}, x, 0 ))</td>
<td></td>
</tr>
<tr>
<td>( q_1 )</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>
<tr>
<td>( q_2 )</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>
<tr>
<td>( q_3 )</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>
<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 of the Turing machine with states and transitions.
Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  \hspace{1em} $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ \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>($q_{\text{yes}}, \square, 0$)</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 b b b x c c c □ □ □
```

**State:** $q_1$
**Turing Machine**

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  \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>

$\text{Z. Sawa (TU Ostrava): Introd. to Theoretical Computer Science}$

February 6, 2014 264 / 401
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_{\text{no}}, a, +1) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_0, a, +1) )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_{\text{no}}, \square, 0) )</td>
<td>( (q_{\text{no}}, a, +1) )</td>
<td>( (q_{\text{no}}, b, +1) )</td>
<td>( (q_{\text{yes}}, 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_{\text{no}}, b, +1) )</td>
<td>( (q_{\text{yes}}, c, 0) )</td>
<td>( (q_2, c, +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_{\text{yes}}, 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, -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>
**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 Diagram](image-url)
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<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>

\( Z. \) Sawa (TU Ostrava) Introd. to Theoretical Computer Science February 6, 2014 264 / 401
**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\text{□}, a, b, c, x\} \]

<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}}, \text{□}, 0 )</td>
<td>( q_{\text{no}}, a, +1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_{\text{no}}, x, 0 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, □, 0 )</td>
<td>( q_{\text{no}}, a, +1 )</td>
<td>( q_{\text{no}}, b, +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}}, □, 0 )</td>
<td>( q_{\text{no}}, □, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, b, +1 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_{\text{no}}, □, −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_{\text{no}}, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_{\text{no}}, □, +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_{\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 \( \Gamma = \{ \sqcup, a, b, c, x \} \)

\[\begin{array}{|c|cccc|}
\hline
\delta & \sqcup & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \sqcup, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \sqcup, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \sqcup, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \sqcup, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \sqcup, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}\]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_{\text{no}}, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_{\text{no}}, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]

\( q_2 \)
Turing Machine

Language \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\( Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \)

\( \Sigma = \{ a, b, c \} \quad \Gamma = \{ \square, a, b, c, x \} \)

\[ \begin{array}{|c|c|c|c|c|c|}
\hline
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array} \]
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\sqcup, a, b, c, x\} \]

<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_{\text{no}}, \sqcup, 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}}, \sqcup, 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>
</tbody>
</table>

Diagram:

- \( q_3 \)
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>($q_{no}, \square, -1$)</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{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_{\text{no}}, \square, +1 )</td>
<td>( q_4, a, -1 )</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 $L = \{ a^n b^n c^n \mid n \geq 0 \}$

$Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \}$ \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>

$q_4$
Turing Machine

Language \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\[ \Sigma = \{ a, b, c \} \quad \Gamma = \{ \square, a, b, c, x \} \]

\[
\begin{array}{|c|ccccc|}
\hline
\delta  & \square & a & b & c & x \\
\hline
q_0     & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1     & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2     & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3     & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4     & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]
### Turing Machine

**Language** \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\[ \Sigma = \{ a, b, c \} \quad \Gamma = \{ \square, a, b, c, x \} \]

<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]

---

Z. Sawa (TU Ostrava) | Introd. to Theoretical Computer Science | February 6, 2014 | 264 / 401
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_{\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>

\( Z. ~ Sawa \) (TU Ostrava)  
Introd. to Theoretical Computer Science  
February 6, 2014  264 / 401
Turing Machine

Language $L = \{a^nb^nc^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\}$  $F = \{q_{yes}, q_{no}\}$

$\Sigma = \{a, b, c\}$  $\Gamma = \{\square, a, b, c, x\}$

<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>

$q_4$
**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>

\[ Z. \text{ Sawa (TU Ostrava)} \quad \text{Introd. to Theoretical Computer Science} \quad \text{February 6, 2014} \quad 264 / 401 \]
**Turing Machine**

**Language** \( L = \{ a^n b^n c^n | n \geq 0 \} \)

- \( Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \)
- \( F = \{ q_{\text{yes}}, q_{\text{no}} \} \)
- \( \Sigma = \{ a, b, c \} \)
- \( \Gamma = \{ \square, a, b, c, x \} \)

<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_{\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_{\text{no}}, b, 0) )</td>
<td>( (q_4, c, -1) )</td>
<td>( (q_4, x, -1) )</td>
</tr>
</tbody>
</table>

- Z. Sawa (TU Ostrava)
- Introd. to Theoretical Computer Science
- February 6, 2014 264 / 401
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 \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \)

\( F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \)

\( \Gamma = \{\blacksquare, a, b, c, x\} \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \blacksquare )</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}}, \blacksquare, 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}}, \blacksquare, 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}}, \blacksquare, 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}}, 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_0, 0) )</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 \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\( Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \)

\( \Sigma = \{ a, b, c \} \quad \Gamma = \{ \Box, a, b, c, x \} \)

\[
\begin{array}{c|cccccc}
\delta & \Box & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \Box, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \Box, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \Box, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_{\text{no}}, \Box, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \Box, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
**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, -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{yes}}, \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:** \( q_4 \)
- **Input Tape:** \( \square \square x x a a x x b b x x c c \square \square \square \)
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$  $\Gamma = \{\square, a, b, c, x\}$

<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>$(q_{\text{no}}, \square, -1)$</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>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_{\text{no}}, x, +1)$</td>
</tr>
</tbody>
</table>

$\begin{array}{cccccccc}
\square & \square & x & x & a & a & x & x & b & b & x & x & c & c & \square & \square & \square
\end{array}$
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$

$F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$

$\Gamma = \{\Box, a, b, c, x\}$

<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 \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<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)  Introd. to Theoretical Computer Science  February 6, 2014  264 / 401
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_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_4, x, -1 ) )</td>
</tr>
</tbody>
</table>
### Turing Machine

**Language** \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

| \( \delta \) | \( \square \) | \( a \) | \( b \) | \( c \) | \( x \) |
|---|---|---|---|---|
| \( q_0 \) | \( (q_{\text{yes}}, \square, 0) \) | \( (q_1, x, +1) \) | \( (q_{\text{no}}, b, 0) \) | \( (q_{\text{no}}, c, 0) \) | \( (q_0, x, +1) \) |
| \( q_1 \) | \( (q_{\text{no}}, \square, 0) \) | \( (q_1, a, +1) \) | \( (q_2, x, +1) \) | \( (q_{\text{no}}, c, 0) \) | \( (q_1, x, +1) \) |
| \( q_2 \) | \( (q_{\text{no}}, \square, 0) \) | \( (q_{\text{no}}, a, 0) \) | \( (q_2, b, +1) \) | \( (q_3, x, +1) \) | \( (q_2, x, +1) \) |
| \( q_3 \) | \( (q_4, \square, -1) \) | \( (q_{\text{no}}, a, 0) \) | \( (q_{\text{no}}, b, 0) \) | \( (q_3, c, +1) \) | \( (q_3, x, +1) \) |
| \( q_4 \) | \( (q_0, \square, +1) \) | \( (q_4, a, -1) \) | \( (q_4, b, -1) \) | \( (q_4, c, -1) \) | \( (q_4, x, -1) \) |

**Diagram**

- \( q_1 \)
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \]

<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 $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  \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>$(q_4, \Box, -1)$</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 $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$

$F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$

$\Gamma = \{\blacksquare, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\blacksquare$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \blacksquare, 0)$</td>
<td>$(q_{\text{no}}, a, +1)$</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>$(q_{\text{no}}, \blacksquare, 0)$</td>
<td>$(q_{\text{no}}, a, +1)$</td>
<td>$(q_{\text{no}}, b, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_1, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \blacksquare, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, c, +1)$</td>
<td>$(q_3, x, +1)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_{\text{no}}, \blacksquare, -1)$</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>$(q_{\text{no}}, \blacksquare, +1)$</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>
The language $L = \{a^n b^n c^n \mid n \geq 0\}$ is defined by the following Turing Machine:

- **States** $Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$
- **Final States** $F = \{q_{\text{yes}}, q_{\text{no}}\}$
- **Input Symbols** $\Sigma = \{a, b, c\}$
- **Tape Symbols** $\Gamma = \{\square, a, b, c, x\}$

The transition function $\delta$ is given as follows:

<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>

The diagram shows the machine transitioning from state $q_0$ to state $q_2$, changing the tape symbols as specified in the transition function.
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>$\square$ $q_{\text{yes}}$, $\square$, 0</th>
<th>$a$ $q_1$, $x$, +1</th>
<th>$b$ $q_{\text{no}}$, $b$, 0</th>
<th>$c$ $q_{\text{no}}$, $c$, 0</th>
<th>$x$ $q_0$, $x$, +1</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>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</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>

- **Diagram**
  - Starting state: \( q_2 \)
  - Transition: \( \square \) to \( q_2 \)
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\#, a, b, c, x\} \)

\[
\begin{array}{c|cccccc}
\delta & \# & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \# , 0) & (q_1, x , +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \# , 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \# , 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \# , -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \# , +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$  $\Gamma = \{\square, a, b, c, x\}$

<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>

$q_3$
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
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>$\square$</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)  Introd. to Theoretical Computer Science  February 6, 2014  264 / 401
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>

Z. Sawa (TU Ostrava) Introd. to Theoretical Computer Science February 6, 2014 264 / 401
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$

$F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$

$\Gamma = \{\square, a, b, c, x\}$

$\delta$

<table>
<thead>
<tr>
<th></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>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_4, c, -1)$</td>
<td>$(q_4, x, -1)$</td>
</tr>
</tbody>
</table>

$q_4$
Turing Machine

Language \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\( Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \)

\( \Sigma = \{ a, b, c \} \quad \Gamma = \{ \square, a, b, c, x \} \)

<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_{\text{no}}, c, +1 )</td>
<td>( q_{\text{no}}, c, +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, -1 )</td>
<td>( q_{\text{no}}, c, -1 )</td>
</tr>
</tbody>
</table>

Z. Sawa (TU Ostrava) | Introd. to Theoretical Computer Science | February 6, 2014 264 / 401
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_{\text{no}}, c, +1 )</td>
<td>( q_{\text{no}}, 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_{\text{no}}, x, -1 )</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\}\)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \) \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>
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>$(q_{\text{no}}, a, 0)$</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_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>

**Diagram**

Input configuration: \( \Box \Box x x x x a x x x x b x x x x c \Box \Box \Box \Box \)

State transition: \( q_4 \)
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\Box, a, b, c, x\} \)

\[\begin{array}{|c|c|c|c|c|}
\hline
\delta & \Box & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \Box, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \Box, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \Box, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \Box, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \Box, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}\]
### Turing Machine

**Language** \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

**States** \( Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \)

**Final States** \( F = \{ q_{\text{yes}}, q_{\text{no}} \} \)

**Alphabets** \( \Sigma = \{ a, b, c \} \)

**Tape Symbols** \( \Gamma = \{ \square, a, b, c, x \} \)

<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_{\text{no}}, a, -1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, -1 )</td>
<td>( q_{\text{no}}, x, -1 )</td>
</tr>
</tbody>
</table>

Diagram:

- \( q_4 \)
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ \hspace{1cm} $F = \{q_{\text{yes}}, q_{\text{no}}\}$

$\Sigma = \{a, b, c\}$ \hspace{1cm} $\Gamma = \{\square, a, b, c, x\}$

$\delta$

<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>

$q_4$
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>$(q_{no}, b, 0)$</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 \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\( Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \) \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_{\text{no}}, x, -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}}, x, -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>( \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>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( (q_0, \Box, +1) )</td>
<td>( (q_4, a, -1) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_4, c, -1) )</td>
<td>( (q_4, x, -1) )</td>
<td></td>
</tr>
</tbody>
</table>

\( q_0 \)
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} $\Gamma = \{\blacksquare, a, b, c, x\}$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\blacksquare$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_{\text{yes}}, \blacksquare, 0)$</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>$(q_{\text{no}}, \blacksquare, 0)$</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>$(q_{\text{no}}, \blacksquare, 0)$</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>$(q_4, \blacksquare, -1)$</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>$(q_0, \blacksquare, +1)$</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 \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\( Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \)

\( \Sigma = \{ a, b, c \} \quad \Gamma = \{ \square, a, b, c, x \} \)

\[
\begin{array}{c|cccccc}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$  \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>$(q_{\text{no}}, b, +1)$</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>

$q_1$
### 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}} )</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}} )</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}} )</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** \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\[ \Sigma = \{ a, b, c \} \quad \Gamma = \{ \&, a, b, c, x \} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( &amp; )</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}}, &amp;, 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}}, &amp;, 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}}, &amp;, 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}}, &amp;, -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}}, &amp;, +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 \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\[ \Sigma = \{ a, b, c \} \quad \Gamma = \{ \Box, a, b, c, x \} \]

<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 $\Gamma = \{\square, a, b, c, x\}$

<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>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_0, x, +1)$</td>
<td></td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, +1)$</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>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_2, x, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_2, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_{\text{yes}}, \square, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, +1)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</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>$(q_{\text{no}}, c, 0)$</td>
</tr>
</tbody>
</table>

$\square$ $\square$ $x$ $x$ $x$ $x$ $x$ $x$ $x$ $x$ $x$ $x$ $x$ $c$ $\square$ $\square$ $\square$
**Turing Machine**

**Language** \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\[ \Sigma = \{ a, b, c \} \quad \Gamma = \{ \square, a, b, c, x \} \]

<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]

---

Z. Sawa (TU Ostrava) Introd. to Theoretical Computer Science February 6, 2014 264 / 401
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}}, □, 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_{\text{no}}, □, -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}}, □, +1)</td>
<td>(q_{\text{no}}, a, -1)</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** \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\[ \Sigma = \{ a, b, c \} \quad \Gamma = \{ \Box, a, b, c, x \} \]

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( q_0 )</th>
<th>( q_1 )</th>
<th>( q_2 )</th>
<th>( q_3 )</th>
<th>( q_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Box )</td>
<td>(q_{\text{yes}}, \Box, 0)</td>
<td>(q_{\text{no}}, \Box, 0)</td>
<td>(q_{\text{no}}, \Box, 0)</td>
<td>(q_{\text{4}}, \Box, -1)</td>
<td>(q_{0}, \Box, +1)</td>
</tr>
<tr>
<td>a</td>
<td>(q_{1}, x, +1)</td>
<td>(q_{2}, x, +1)</td>
<td>(q_{no}, a, 0)</td>
<td>(q_{no}, a, 0)</td>
<td>(q_{4}, a, -1)</td>
</tr>
<tr>
<td>b</td>
<td>(q_{no}, b, 0)</td>
<td>(q_{no}, c, 0)</td>
<td>(q_{3}, x, +1)</td>
<td>(q_{3}, c, +1)</td>
<td>(q_{4}, b, -1)</td>
</tr>
<tr>
<td>c</td>
<td>(q_{no}, c, 0)</td>
<td>(q_{no}, c, 0)</td>
<td>(q_{2}, x, +1)</td>
<td>(q_{3}, x, +1)</td>
<td>(q_{4}, c, -1)</td>
</tr>
<tr>
<td>x</td>
<td>(q_{0}, x, +1)</td>
<td>(q_{1}, x, +1)</td>
<td>(q_{2}, x, +1)</td>
<td>(q_{3}, x, +1)</td>
<td>(q_{4}, x, -1)</td>
</tr>
</tbody>
</table>

**Diagram:**

- Start state: \( q_0 \)
- \( q_2 \) is the current state.
- Move right and change symbols according to the transition table.
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\}$

$F = \{q_{yes}, q_{no}\}$

$\Sigma = \{a, b, c\}$

$\Gamma = \{\square, a, b, c, x\}$

<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>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_{no}, x, +1)$</td>
<td></td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{no}, \square, 0)$</td>
<td>$(q_{no}, a, +1)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_{no}, x, +1)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{no}, \square, 0)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_{no}, b, +1)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_{no}, x, +1)$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_{no}, \square, -1)$</td>
<td>$(q_{no}, a, 0)$</td>
<td>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, +1)$</td>
<td>$(q_{no}, x, +1)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>$(q_{no}, \square, +1)$</td>
<td>$(q_{no}, a, -1)$</td>
<td>$(q_{no}, b, -1)$</td>
<td>$(q_{no}, c, -1)$</td>
<td>$(q_{no}, x, -1)$</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

\[
\begin{array}{l|cccc|}
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
### 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{no}}, \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_4, c, -1) )</td>
<td>( (q_4, x, -1) )</td>
</tr>
</tbody>
</table>

### Diagram

```
□ □ x x x x x x x x x x x x □ □ □
```

**State**: \( q_4 \)
Turing Machine

Language $L = \{ a^n b^n c^n \mid n \geq 0 \}$

$Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \}$

$\Sigma = \{ a, b, c \} \quad \Gamma = \{ \square, a, b, c, x \}$

<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>

$Z. Sawa (TU Ostrava) \quad \text{Introd. to Theoretical Computer Science} \quad \text{February 6, 2014} \quad 264 / 401$
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>

Z. Sawa (TU Ostrava) | Introd. to Theoretical Computer Science | February 6, 2014 264 / 401
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} $\Gamma = \{\sqcup, a, b, c, x\}$

<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>
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>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, c, 0)$</td>
<td>$(q_{\text{no}}, x, +1)$</td>
<td></td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, a, +1)$</td>
<td>$(q_{\text{no}}, a, +1)$</td>
<td></td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, +1)$</td>
<td>$(q_{\text{no}}, b, +1)$</td>
<td></td>
</tr>
<tr>
<td>$q_3$</td>
<td>$(q_{\text{no}}, \square, 0)$</td>
<td>$(q_{\text{no}}, a, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td>$(q_{\text{no}}, b, 0)$</td>
<td></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>$(q_{\text{no}}, b, 0)$</td>
<td></td>
</tr>
</tbody>
</table>

$q_4$
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\boxempty, a, b, c, x\} \)

\[
\begin{array}{c|cccccc}
\delta & \boxempty & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \boxempty, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \boxempty, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \boxempty, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \boxempty, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \boxempty, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
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_{\text{no}}, a, +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_{\text{no}}, a, +1) )</td>
<td>( (q_{\text{no}}, b, +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_{\text{no}}, b, +1) )</td>
<td>( (q_{\text{no}}, c, 0) )</td>
<td>( (q_{\text{no}}, x, +1) )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( (q_{\text{no}}, a, -1) )</td>
<td>( (q_{\text{no}}, a, +1) )</td>
<td>( (q_{\text{no}}, b, 0) )</td>
<td>( (q_{\text{no}}, c, +1) )</td>
<td>( (q_{\text{no}}, x, +1) )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( (q_{\text{no}}, a, -1) )</td>
<td>( (q_{\text{no}}, a, +1) )</td>
<td>( (q_{\text{no}}, a, 0) )</td>
<td>( (q_{\text{no}}, c, -1) )</td>
<td>( (q_{\text{no}}, x, -1) )</td>
</tr>
</tbody>
</table>
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ \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>

$\text{Z. Sawa (TU Ostrava)}$
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_{\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>
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 \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \]

\[ \Sigma = \{a, b, c\} \quad \quad \Gamma = \{\square, a, b, c, x\} \]

<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_{\text{no}}, 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_{\text{no}}, b, +1 )</td>
<td>( q_{\text{no}}, x, +1 )</td>
<td>( q_{\text{no}}, 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_{\text{no}}, c, +1 )</td>
<td>( q_{\text{no}}, x, +1 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_{\text{no}}, \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_{\text{no}}, x, -1 )</td>
</tr>
</tbody>
</table>
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\}$ \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 \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

<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, x, +1 )</td>
<td>( q_4, a, +1 )</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>
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>($q_{no}, b, 0$)</td>
<td>($q_{no}, c, 0$)</td>
<td>($q_{no}, b, 0$)</td>
</tr>
</tbody>
</table>

\[\begin{array}{cccccccccccccccc}
\square & \square & x & x & x & x & x & x & x & x & x & x & x & \square & \square & \square \\
\end{array}\]
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_{\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_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_{\text{no}}, \square, +1 )</td>
<td>( q_4, a, -1 )</td>
<td>( q_{\text{no}}, b, 0 )</td>
<td>( q_4, c, -1 )</td>
<td>( q_4, x, -1 )</td>
</tr>
</tbody>
</table>

\[ Z. \ Sawa \ (TU \ Ostrava) \]

Introd. to Theoretical Computer Science

February 6, 2014 264 / 401
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>$(q_{\text{no}}, \square, -1)$</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>
# 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**: \( \Sigma = \{ a, b, c \} \)
- **Notification Symbols**: \( \Gamma = \{ \square, a, b, c, x \} \)

<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>

![Diagram](image_url)
Turing Machine

Language \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\( Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \)
\( F = \{ q_{\text{yes}}, q_{\text{no}} \} \)
\( \Sigma = \{ a, b, c \} \)
\( \Gamma = \{ \Box, a, b, c, x \} \)

\[
\begin{array}{c|cccccc}
\delta & \Box & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \Box, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \Box, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \Box, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_{\text{no}}, \Box, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \Box, +1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_4, c, -1) & (q_4, x, -1) \\
\end{array}
\]
Turing Machine

Language \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\[ \Sigma = \{ a, b, c \} \quad \Gamma = \{ \square, a, b, c, x \} \]

\[
\begin{array}{|c|c|c|c|c|c|c|}
\hline
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]
Turing Machine

Language $L = \{a^n b^n c^n \mid n \geq 0\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_{yes}, q_{no}\}$ \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>$(q_{no}, b, 0)$</td>
<td>$(q_{no}, c, 0)$</td>
<td>$(q_{no}, x, +1)$</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>$(q_{no}, c, 0)$</td>
<td>$(q_{no}, x, +1)$</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 $\Gamma = \{\Box, a, b, c, x\}$

$\delta$

<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>$(q_{\text{no}}, \Box, 1)$</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 \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\( Q = \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \quad F = \{q_{\text{yes}}, q_{\text{no}}\} \)

\( \Sigma = \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \)

<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>

\( q_0 \)
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_{\text{no}}, b, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
<td>( q_{\text{no}}, c, 0 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_{\text{no}}, \square, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</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}}, a, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
<td>( q_{\text{no}}, a, 0 )</td>
</tr>
</tbody>
</table>
Turing Machine

Language \( L = \{a^n b^n c^n \mid n \geq 0\} \)

\[ \begin{align*} Q &= \{q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}}\} \\ F &= \{q_{\text{yes}}, q_{\text{no}}\} \\ \Sigma &= \{a, b, c\} \quad \Gamma = \{\square, a, b, c, x\} \end{align*} \]

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\delta & \square & a & b & c & x \\
\hline
q_0 & (q_{\text{yes}}, \square, 0) & (q_1, x, +1) & (q_{\text{no}}, b, 0) & (q_{\text{no}}, c, 0) & (q_0, x, +1) \\
q_1 & (q_{\text{no}}, \square, 0) & (q_1, a, +1) & (q_2, x, +1) & (q_{\text{no}}, c, 0) & (q_1, x, +1) \\
q_2 & (q_{\text{no}}, \square, 0) & (q_{\text{no}}, a, 0) & (q_2, b, +1) & (q_3, x, +1) & (q_2, x, +1) \\
q_3 & (q_4, \square, -1) & (q_{\text{no}}, a, 0) & (q_{\text{no}}, b, 0) & (q_3, c, +1) & (q_3, x, +1) \\
q_4 & (q_0, \square, +1) & (q_4, a, -1) & (q_4, b, -1) & (q_4, c, -1) & (q_4, x, -1) \\
\hline
\end{array}
\]
**Turing Machine**

**Language** \( L = \{ a^n b^n c^n \mid n \geq 0 \} \)

\[ Q = \{ q_0, q_1, q_2, q_3, q_4, q_{\text{yes}}, q_{\text{no}} \} \quad \quad F = \{ q_{\text{yes}}, q_{\text{no}} \} \]

\[ \Sigma = \{ a, b, c \} \quad \quad \Gamma = \{ \Box, a, b, c, x \} \]

<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_{\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_{\text{yes}}, \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 – Multiplication by Three

0, 1; +
0; −
1; −
1 → 0; −

0, □ → 1; −
0, □ → 0; −

□ □ □ 1 1 1 0 0 1 1 □ □ □ □

R

Z. Sawa (TU Ostrava) Introd. to Theoretical Computer Science February 6, 2014 266 / 401
Turing Machine – Multiplication by Three

0, 1; +

0; −

1; −

1 → 0; −

0, 0 → 1; −

0, 0 → 0; −

□; −

□; −
Turing Machine – Multiplication by Three

0, 1; +

0; –

1; –

1 → 0; –

0, ⊖ → 1; –

0, ⊖ → 0; –

P

R

[Diagram of a Turing Machine with states and transitions]
Turing Machine – Multiplication by Three

[Diagram of a Turing machine with states R, 0, 1, 2, and transitions labeled with symbols and operations like 0, 1; +, 0; −, 1; −, 1 → 0; −, 0, □ → 1; −, 0, □ → 0; −.]
Turing Machine – Multiplication by Three

- **Initial State**: Turing machine starts with a blank tape and a marker at the leftmost position.
- **States**: The machine moves through states R (reading), 0, 1, and 2.
- **Transitions**: The machine reads a symbol (0 or 1) and depending on the current state, performs an operation and moves to the next state.

**Transitions**:
- From R to 0: Write 0, move left.
- From 0 to 1: Write 1, move right.
- From 1 to 2: Replace symbol with another symbol, move right.
- From 2 to R: Replace symbol with another symbol, move left.

**Tape**:
- The tape is initially blank and contains symbols that the machine processes.

**Final State**: The machine reaches a final state to halt.

---

Z. Sawa (TU Ostrava)  Introd. to Theoretical Computer Science  February 6, 2014  266 / 401
Turing Machine – Multiplication by Three

0,1; +

0; −

1; −

1 → 0; −

0, □ → 1; −

0, □ → 0; −

R

P

Z. Sawa (TU Ostrava) Introd. to Theoretical Computer Science February 6, 2014 266 / 401
Turing Machine – Multiplication by Three

R

0, 1; +

0; -

1; -

1 \rightarrow 0; -

1; -

0, 1 1 1 0 0 1 1

0, \square \rightarrow 1; -

0, \square \rightarrow 0; -

P

0; -

\square; -

\square; -

\square; -

\square; -

R

Z. Sawa (TU Ostrava)

Introd. to Theoretical Computer Science

February 6, 2014 266 / 401
Turing Machine – Multiplication by Three

0, 1; +
0; –
1; –
1 \rightarrow 0; –

0, □ \rightarrow 1; –
0, □ \rightarrow 0; –

□ □ □ 1 1 1 0 0 1 1 □ □ □

□ □ □ □ 1 1 1 0 0 1 1 □ □ □

0
Turing Machine – Multiplication by Three

0, 1; +

0; −

1; −

1 → 0; −

Z. Sawa (TU Ostrava) Introd. to Theoretical Computer Science February 6, 2014 266 / 401
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; − 1; −

R 0 1 2

Z. Sawa (TU Ostrava) Introd. to Theoretical Computer Science February 6, 2014 266 / 401
Turing Machine – Multiplication by Three

0, 1; +

0; −

1; −

1 → 0; −

0, □ → 1; −

0, □ → 0; −

0
Turing Machine – Multiplication by Three
Turing Machine – Multiplication by Three

0, 1; +

0; −

1; −

0, 0; → 1; −

0, 1; → 0; −

1; −

P

R

1

2

[Diagram of a Turing machine with states and transitions labeled with symbols and numbers.]
Turing Machine – Multiplication by Three

Z. Sawa (TU Ostrava)
Introd. to Theoretical Computer Science
February 6, 2014 266 / 401
Turing Machine – Multiplication by Three

0, 1; +

0; −

1; −

1 → 0; −

0, □ → 1; −

0, □ → 0; −

□; −

□; −

□; −
Turing Machine – Multiplication by Three

0, 1; +
0; −
1; −

0, □ → 1; −
0, □ → 0; −

1 → 0; −
Turing Machine – Multiplication by Three

0, 1; +

0; −

1; −

1 → 0; −

0, □ → 1; −

0, □ → 0; −

P
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.
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.
Church-Turing Thesis

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.
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 would like to compare different algorithms and choose a better one.

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.
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    2
JZERO   output
SUB     3
JZERO   found
load:   LOAD    2
ADD     =1
JUMP    loop
found:  LOAD    2
output: WRITE
HALT
```
Time Complexity

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
Time Complexity

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
Time Complexity

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
Time Complexity

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:  9
         Cell 4:  0

Output: Instructions: 2
Time Complexity

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
Time Complexity

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
Time Complexity

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
### Time Complexity

```
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:**

```
Cell 0:  13
Cell 1:  0
Cell 2:  1
Cell 3:  9
```

**Output:**

```
Instructions: 6
```
Time Complexity

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
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
Time Complexity

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
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
Time Complexity

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
Time Complexity

\[
\begin{align*}
\text{start:} & \quad \text{READ} \\
& \quad \text{STORE} \quad 3 \\
& \quad \text{LOAD} \quad =1 \\
\text{loop:} & \quad \text{STORE} \quad 2 \\
& \quad \text{READ} \\
& \quad \text{JZERO} \quad \text{output} \\
& \quad \text{SUB} \quad 3 \\
& \quad \text{JZERO} \quad \text{found} \\
& \quad \text{LOAD} \quad 2 \\
& \quad \text{ADD} \quad =1 \\
& \quad \text{JUMP} \quad \text{loop} \\
\text{found:} & \quad \text{LOAD} \quad 2 \\
\text{output:} & \quad \text{WRITE} \\
& \quad \text{HALT}
\end{align*}
\]

Input:
9, 13, 5, 9, 7, 2, 0

Cell 0: 2
Cell 1: 0
Cell 2: 2
Cell 3: 9
Cell 4: 0

Output:

Instructions: 12
Time Complexity

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
Time Complexity

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: 14
Time Complexity

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
Time Complexity

\begin{verbatim}
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
\end{verbatim}
Time Complexity

```
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
Time Complexity

```
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
Time Complexity

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
Time Complexity

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
Time Complexity

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

Input:
Cell 0: 9
Cell 1: 0
Cell 2: 3
Cell 3: 9
Cell 4: 0

Output:
Instructions: 21
Time Complexity

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
Time Complexity

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
Time Complexity

```
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
Time Complexity

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
Time Complexity

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
Time Complexity

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
In this example, where the input is

9, 13, 5, 9, 7, 2, 0

the algorithm performed 27 instructions.
In this example, where the input is 

\[ 9, 13, 5, 9, 7, 2, 0 \]

the algorithm performed 27 instructions.

We can try to analyze how many instructions the algorithm performs in general for an input 

\[ x, a_1, a_2, \ldots, a_n, 0 \]
In this example, where the input is

\[ 9, 13, 5, 9, 7, 2, 0 \]

the algorithm performed 27 instructions.

We can try to analyze how many instructions the algorithm performs in general for an input

\[ x, a_1, a_2, \ldots, a_n, 0 \]

If for all \( i \) (where \( 1 \leq i \leq n \)) is \( x \neq a_i \):

\[ 3 + 8 \cdot n + 3 + 2 = 8n + 8 \]
In this example, where the input is \(9, 13, 5, 9, 7, 2, 0\), the algorithm performed 27 instructions.

We can try to analyze how many instructions the algorithm performs in general for an input \(x, a_1, a_2, \ldots, a_n, 0\).

- If for all \(i\) (where \(1 \leq i \leq n\)) is \(x \neq a_i\):
  \[
  3 + 8 \cdot n + 3 + 2 = 8n + 8
  \]

- If for some \(i\) (where \(1 \leq i \leq n\)) is \(x = a_i\):
  \[
  3 + 8 \cdot (i - 1) + 5 + 3 = 8i + 3
  \]
Size of Input

For different input data the program performs a different number of instructions.

If we want to analyze somehow the number of performed instructions, it is useful to introduce the notion of the size of an input.

Typically, the size of an input is a number specifying how “big” is the given instance (a bigger number means a bigger instance).

**Example:** For problem “Searching” where all inputs are of the form

\[ x, a_1, a_2, \ldots, a_n, 0 \]

we can choose the number \( n \) as the size of the input.

This way the size of the input \( 9, 13, 5, 9, 7, 2, 0 \) is 5.
**Remark:** We can define the size of an input as we like depending on what is useful for our analysis.

The size of an input is not strictly determinable but there are usually some natural choices based on the nature of the problem.

**Examples:**

- For the problem “Sorting”, where the input is a sequence of numbers $a_1, a_2, \ldots, a_n$ and the output the same sequence sorted, we can take $n$ as the size of the input.

- For the problem “Primality” where the input is a natural number $x$ and where the question is whether $x$ is a prime, we can take the number of bits of the number $x$ as the size of the input. (The other possibility is to take directly the value $x$ as the size of the input.)
Sometimes it is useful to describe the size of an input with several numbers.

For example for problems where the input is a graph, we can define the size of the input as a pair of numbers $n, m$ where:

- $n$ – the number of nodes of the graph
- $m$ – the number of edges of the graph

**Remark:** The other possibility is to define the size of the input as one number $n + m$. 

Size of Input

In general, we can define the size of an input for an arbitrary problem as follows:

- When the input is a word over some alphabet $\Sigma$: the length of word $w$

- When the input is as a sequence of bits (i.e., a word over $\{0, 1\}$): the number of bits in this sequence

- When the input is a natural number $x$: the number of bits in the binary representation of $x$
Time Complexity

We want to analyze a particular algorithm (its particular implementation).

We want to know how many steps the algorithm performs when it gets an input of size 1, 2, 3, 4, . . .

It is obvious that even for inputs of the same size the number of performed steps can be different.

Let us assume that $X$ is the set of all possible inputs of the given problem, and $g(x)$ is the number of steps performed by the algorithm for an input $x \in X$.

Let us denote the size of input $x \in X$ as $|x|$.  

Now we define a function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that for $n \in \mathbb{N}$ is

$$f(n) = \max \{ g(x) \mid x \in X, \ |x| = n \}$$
Time Complexity in the Worst Case

Such function $f(n)$ (i.e., a function that for the given algorithm and the given definition of the size of an input assigns to every natural number $n$ the maximal number of instructions performed by the algorithm if it obtains an input of size $n$) is called the **time complexity of the algorithm in the worst case**.
Time Complexity in the Worst Case

Such function \( f(n) \) (i.e., a function that for the given algorithm and the given definition of the size of an input assigns to every natural number \( n \) the maximal number of instructions performed by the algorithm if it obtains an input of size \( n \)) is called the \textit{time complexity of the algorithm in the worst case}.

Example: As we found out, the previously described algorithm solving the problem “Searching” performs for an input \( x, a_1, a_2, \ldots, a_n, 0 \) the following number of steps:

- If for all \( i \) it holds that \( x \neq a_i \), then the algorithm performs \( 8n + 8 \) steps.
- If for some \( i \) it holds that \( x = a_i \), then the algorithm performs \( 8i + 3 \) steps.

Since in the second case it is always \( i \leq n \), the time complexity of the algorithm in the worst case is \( f(n) = 8n + 8 \).
Sometimes it make sense to analyze the time complexity in an average case.

In this case, we do not define $f(n)$ as the maximum but as the arithmetic mean of the set

$$\{g(x) \mid x \in X, |x| = n\}$$

- It is usually more difficult to determine the time complexity in an average case than to determine the time complexity in the worst case.
- Often, these two function are not very different but sometimes the difference is significant.

**Remark:** It usually makes no sense to analyze the time complexity in the best case.
A program works on an input of size $n$. Let us assume that for an input of size $n$, the program performs $f(n)$ operations and that an execution of one operation takes $1\mu s$ ($10^{-6} s$).

<table>
<thead>
<tr>
<th>$f(n)$</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>
Growth of Functions

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.

<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>
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

- It is usually quite difficult to express the complexity exactly.

- The exact complexity depends on the used model of computation and on the particular implementation (on details of this implementation).

- We are interested in the complexity for big inputs. For small inputs usually even nonefficient algorithms work fast.

- We usually do not need to know the exact number of performed instructions and we will be satisfied with some estimation of how fast this number grows when the size of an input grows.

- So we use the so called **asymptotic notation**, which allows us to ignore unimportant details and to estimate approximately how fast the given function grows. This simplifies the analysis considerably.
Asymptotic Notation

Let us take an arbitrary function $g : \mathbb{N} \rightarrow \mathbb{N}$. Expressions $O(g)$, $\Omega(g)$, and $\Theta(g)$ denote sets of functions of the type $\mathbb{N} \rightarrow \mathbb{N}$, where:

- $O(g)$ – the set of all functions that grow at most as fast as $g$
- $\Omega(g)$ – the set of all functions that grow at least as fast as $g$
- $\Theta(g)$ – the set of all functions that grow as fast as $g$

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”
Definition

Let us consider an arbitrary function $g : \mathbb{N} \to \mathbb{N}$. For a function $f : \mathbb{N} \to \mathbb{N}$ we have $f \in O(g)$ iff

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0) : f(n) \leq c \cdot g(n).$$
Asymptotic Notation – Symbol $\Omega$

Definition

Let us consider an arbitrary function $g : \mathbb{N} \rightarrow \mathbb{N}$. For a function $f : \mathbb{N} \rightarrow \mathbb{N}$ we have $f \in \Omega(g)$ iff

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0) : c \cdot g(n) \leq f(n).$$
Asymptotic Notation – Symbol $\Theta$

Definition

Let us consider an arbitrary function $g : \mathbb{N} \to \mathbb{N}$. For a function $f : \mathbb{N} \to \mathbb{N}$ we have $f \in \Theta(g)$ iff

$$(\exists c_1 > 0)(\exists c_2 > 0)(\exists n_0 \geq 0)(\forall n \geq n_0) : c_1 g(n) \leq f(n) \leq c_2 g(n).$$
Asymptotic Notation

For simplicity, we consider only functions of type \( \mathbb{N} \to \mathbb{N} \) in the previous definitions.

In fact, these definitions could be extended to all \textit{asymptotically nonnegative} functions of type \( \mathbb{R}_+ \to \mathbb{R} \), which moreover can be undefined on some finite subinterval of its domain.

Function \( f : \mathbb{R}_+ \to \mathbb{R} \) is \textit{asymptotically nonnegative} if it satisfies:

\[
(\exists n_0 \geq 0)(\forall n \geq n_0)(f(n) \geq 0)
\]

**Remark:** For \( n < n_0 \), the value of \( f(n) \) can be undefined.

\( \mathbb{R}_+ = \{ x \in \mathbb{R} \mid x \geq 0 \} \)
Asymptotic Notation

Examples:

\begin{align*}
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)
\end{align*}
Asymptotic Notation

For any function $g : \mathbb{N} \rightarrow \mathbb{N}$ we have:

$$g \in O(g) \quad g \in \Omega(g) \quad g \in \Theta(g)$$

For any pair of functions $f, g : \mathbb{N} \rightarrow \mathbb{N}$ we have:

- $f \in O(g)$ iff $g \in \Omega(f)$
- $f \in \Theta(g)$ iff $g \in \Theta(f)$
- $f \in \Theta(g)$ iff $f \in O(g)$ and $f \in \Omega(g)$

For any functions $f, g, h : \mathbb{N} \rightarrow \mathbb{N}$ 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 $f \in \Theta(h)$
There are pairs of functions $f, g : \mathbb{N} \to \mathbb{N}$ such that

$$f \not\in O(g) \quad \text{and} \quad g \not\in O(f),$$

for example

$$f(n) = n \quad \quad g(n) = n^{1+\sin(n)}.$$

$O(1)$ denotes 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 $f, g : \mathbb{N} \rightarrow \mathbb{N}$ we have:

- $\max(f, g) \in \Theta(f + g)$
- If $f \in O(g)$ then $f + g \in \Theta(g)$

For any functions $f_1, f_2, g_1, g_2 : \mathbb{N} \rightarrow \mathbb{N}$ 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 $f(n) \in O(c^{n^k})$ 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

For any $k, \ell > 0$, such that $k < \ell$, and any $c > 1$ we have:

\[
\begin{align*}
n^k &\in O(n^\ell) & n^\ell &\notin O(n^k) \\
\log_c^k n &\in O(\log_c^\ell n) & \log_c^\ell n &\notin O(\log_c^k n) \\
c^{n^k} &\in O(c^{n^\ell}) & c^{n^\ell} &\notin O(c^{n^k})
\end{align*}
\]

For any $k, \ell > 0$ and any $c > 1$ we have:

\[
\begin{align*}
\log_c^k n &\in O(n^\ell) & n^\ell &\notin O(\log_c^k n) \\
n^k &\in O(c^{n^\ell}) & c^{n^\ell} &\notin O(n^k)
\end{align*}
\]

Remark: $\log_c^k n$ is a more succinct notation for $(\log_c n)^k$.
Asymptotic Notation

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 \( \log_b n = \log_a n \cdot \log_b n \), from which the above mentioned conclusion follows directly.

So for any constants \( a, b > 1 \) we have \( \log_a n \in \Theta(\log_b n) \).

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 \( \Theta(n \log n) \).
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 $f \in O(g)$. 
The asymptotic notation can be straightforwardly generalized to functions of several arguments:

Let us consider, for example, a function $g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$. For function $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ we have $f(n, m) \in O(g(n, m))$ iff

$$(\exists c > 0)(\exists n_0 \geq 0)(\exists m_0 \geq 0)(\forall n \geq n_0)(\forall m \geq m_0)(f(n, m) \leq c g(n, m))$$

In addition to expressions $O(g)$, $\Omega(g)$, and $\Theta(g)$, expressions $o(g)$ and $\omega(g)$ are also sometimes used.

- $o(g)$ – the set of all functions that grow slower than function $g$
- $\omega(g)$ – the set of all functions that grow faster than function $g$

We will not discuss their exact definitions and we will not deal with these expressions any further.
Let us say we would like to analyze the time complexity \( t(n) \) of some algorithm consisting of instructions \( l_1, l_2, \ldots, l_k \):

- If \( m_1, m_2, \ldots, m_k \) are the number of executions of individual instructions for some input \( x \) (i.e., the instruction \( l_i \) is performed \( m_i \) times for the input \( x \)), then the total number of executed instructions for input \( x \) is
  \[
  m_1 + m_2 + \cdots + m_k.
  \]

- Let us consider functions \( f_1, f_2, \ldots, f_k \), where \( f_i : \mathbb{N} \to \mathbb{N} \), and where \( f_i(n) \) is the maximum of numbers of executions of instruction \( l_i \) 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

$$t \in O(f_i).$$

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

$$t \in \Theta(f_i).$$
Example: In the analysis of the complexity of the searching of a number in a sequence we obtained

\[ f(n) = 8n + 8. \]

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 the only cycle, which is performed at most \( n \) times, and there are some inputs, for which it is actually performed \( n \) times.

- 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 of 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:

**Insertion-Sort**($A, n$):

1. for $j := 2$ to $n$ do
   - $x := A[j]$
   - $i := j - 1$
   - while $i > 0$ and $A[i] > x$ do
     - $i := i - 1$
   - end while
   - $A[i + 1] := x$
2. end for

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 $Θ(t(n))$. 
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 accurately computed the total number of executions of the cycle `while`.

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 \( \lfloor n/2 \rfloor \) times in the last \( \lfloor n/2 \rfloor \) iterations of the cycle **for**.

So the cycle **while** is performed at least \( \lfloor n/2 \rfloor \cdot \lceil n/2 \rceil \) times for some inputs.

\[
\lfloor n/2 \rfloor \cdot \lceil n/2 \rceil \geq (n/2 - 1) \cdot (n/2) = \frac{1}{4} n^2 - \frac{1}{2} n
\]

This implies \( t \in \Omega(n^2) \).
Complexity of Algorithms

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.
So far we have considered only the case when the execution of all instructions takes the same time.

In practice, some of instructions can be more time consuming than others.

If we know the times of an execution for each instruction and if we can count the number of their executions, we can compute the overall running time as

$$
\sum_{i} n_i t_i
$$

where $n_i$ is the number of executions of instruction $i$ and $t_i$ is the time of one execution of $i$. 
Let us assume that we analyze the time complexity of some algorithm implemented as a RAM.

So far, we have counted only the number of executed instructions in our analysis. This is known as using the so called uniform-cost measurement.

Estimations of the time complexity using the uniform-cost measurement correspond to the running time on real computers under the assumption that operations, performed by the RAM, can be performed by a real computer in a constant time.

This assumption holds, if numbers, the algorithm works with, are small (they can be stored, say, to 32 or 64 bits).

If the RAM works with “big” numbers (e.g., 1000 bit), the estimation using the uniform-cost measurement will be unrealistic in the sense that a computation on a real computer will take much more time.
To analyse the time complexity of algorithms working with big numbers, we usually use so called \textit{logarithmic-cost measurement}, where a duration of one instruction is not 1 but is proportional to the number of \textbf{bit operations}, which are necessary for an execution of this instruction.

The duration of an execution of an instruction depends on the actual values of its operands.

For example, a duration of an execution of instructions \texttt{ADD} and \texttt{SUB} is equal to the sum of the numbers of bits of their operands.

The duration of an execution of instructions \texttt{MUL} and \texttt{DIV} is equal to the sum of the numbers of bits of their operands.
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.

**Amount of memory** of a RAM $M$ used for an input $x$ is the number of memory cells that are used by $M$ during its computation on $x$.

**Definition**

A **space complexity** of a RAM $M$ (in the worst case) is the function $s : \mathbb{N} \rightarrow \mathbb{N}$, where $s(n)$ is the maximal amount of memory used by $M$ for inputs of size $n$. 
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 a RAM uses at most one cell beside the accumulator in each step).
Complexity of Problems
Complexity of Problems

- It seems that different (algorithmic) problems are of different difficulty.

- More difficult are those problems that require more time and space to be solved.

- We would like to analyze somehow the difficultness of problems
  - absolutely – how much time and space do we need for their solution,
  - relatively – by how much is their solution harder or simpler with respect to other problems.

- Why do we not succeed in finding efficient algorithms for some problems?
  Can there exist an efficient algorithm for a given problem?

- What are practical boundaries of what can be achieved?
It is necessary to distinguish between a complexity of an algorithm and a complexity of a problem.

If we for example study the time complexity in the worst case, informally we could say:

- **complexity of an algorithm** – a function expressing how many steps at most the given algorithm performs for an input of size $n$
- **complexity of a problem** – what is the time complexity of the “most efficient” algorithm for the given problem

A formal definition of a notion “complexity of a problem” in the above sense leads to some technical difficulties. So the notion “complexity of a problem” is not defined as such but it is bypassed by a definition of complexity classes.
Complexity classes are subsets of the set of all (algorithmic) problems. A certain particular complexity class is always characterized by a property that is shared by all the problems belonging to the class.

A typical example of such a property is a property that for the given problem there exists some algorithm with some restrictions (e.g., on its time or space complexity):

- Only a problem for which such algorithm exists belongs to the given class.
- A problem for which such algorithm does not exist does not belong to the class.
Complexity Classes

**Definition**

For every function $f : \mathbb{N} \rightarrow \mathbb{N}$ we define $T(f(n))$ as the class containing exactly those problems for which there exists an algorithm with time complexity $O(f(n))$.

**Example:**

- $T(n)$ – the class of all problems for which there exists an algorithm with time complexity $O(n)$
- $T(n^2)$ – the class of all problems for which there exists an algorithm with time complexity $O(n^2)$
- $T(n \log n)$ – the class of all problems for which there exists an algorithm with time complexity $O(n \log n)$
Complexity Classes

Definition

For every function \( f : \mathbb{N} \rightarrow \mathbb{N} \) we define \( S(f(n)) \) as the class containing exactly those problems for which there exists an algorithm with space complexity \( O(f(n)) \).

Example:

- \( S(n) \) – the class of all problems for which there exists an algorithm with space complexity \( O(n) \)
- \( S(n^2) \) – the class of all problems for which there exists an algorithm with space complexity \( O(n^2) \)
- \( S(n \log n) \) – the class of all problems for which there exists an algorithm with space complexity \( O(n \log n) \)
Remark:

Note that for classes $T(f)$ and $S(f)$ it depends which problems belong to the class on the used computational model (if it is a RAM, a one-tape Turing machine, a multitape Turing machine, . . . ).
Using classes $\mathcal{T}(f(n))$ and $\mathcal{S}(f(n))$ we can define classes \textsc{PTIME} and \textsc{PSPACE} as

$$\text{PTIME} = \bigcup_{k \geq 0} \mathcal{T}(n^k) \quad \quad \text{PSPACE} = \bigcup_{k \geq 0} \mathcal{S}(n^k)$$

- \textbf{PTIME} is the class of all problems for which there exists an algorithm with polynomial time complexity, i.e., with time complexity $O(n^k)$ where $k$ is a constant.

- \textbf{PSPACE} is the class of all all problems for which there exists an algorithm with polynomial space complexity, i.e., with space complexity $O(n^k)$ where $k$ is a constant.
Remark: Since all (reasonable) computational models are able to simulate each other in such a way that in this simulation the number of steps does not increase more than polynomially, the definitions of classes \(\text{PTIME}\) and \(\text{PSPACE}\) are not dependent on the used computational model. For their definition we can use any computational model.

We say that these classes are **robust** – their definitions do not depend on the used computational model.
Other classes are introduced analogously:

**EXPTIME** – the set of all problems for which there exists an algorithm with time complexity $2^{O(n^k)}$ where $k$ is a constant

**EXPSPACE** – the set of all problems for which there exists an algorithm with space complexity $2^{O(n^k)}$ where $k$ is a constant

**LOGSPACE** – the set of all problems for which there exists an algorithm with space complexity $O(\log n)$

**Remark:** Instead of $2^{O(n^k)}$ we can also write $O(c^{n^k})$ where $c$ and $k$ are constants.
If a Turing machine performs $m$ steps then it visits at most $m$ cells on the tape.

This means that if there exists an algorithm for some problem with time complexity $O(f(n))$, the space complexity of this algorithm is (at most) $O(f(n))$.

So it is obvious that the following relationship holds.

**Observation**

For every function $f : \mathbb{N} \rightarrow \mathbb{N}$ is $T(f(n)) \subseteq S(f(n))$.

**Remark:** We can analogously reason in the case of a RAM.
Based on the previous, we see that:

\[
\text{PTIME} \subseteq \text{PSPACE} \\
\text{EXPTIME} \subseteq \text{EXPSPACE}
\]

Since polynomial functions grow more slowly than exponential, we obviously have:

\[
\text{PTIME} \subseteq \text{EXPTIME} \\
\text{LOGSPACE} \subseteq \text{PSPACE} \subseteq \text{EXPSPACE}
\]
For every pair of real numbers $0 \leq \epsilon_1 < \epsilon_2$ is

$$S(n^{\epsilon_1}) \subsetneq S(n^{\epsilon_2})$$

**LOGSPACE \subsetneq PSPACE**

**PSPACE \subsetneq EXPTSPACE**

For every pair of real numbers $0 \leq \epsilon_1 < \epsilon_2$ is

$$T(n^{\epsilon_1}) \subsetneq T(n^{\epsilon_2})$$

**PTIME \subsetneq EXPTIME**
For analyzing relationships between complexity classes it is useful to consider **configurations**.

A configuration is a global state of a machine during one step of a computation.

- For a Turing machine, a configuration is given by the state of its control unit, the content of the tape (resp. tapes), and the position of the head (resp. heads).
- For a RAM, a configuration is given by the content of the memory, by the content of all registers (including IP), by the content of the input and output tapes, and by positions of their heads.
Relationships between Complexity Classes

It should be clear that configurations (or rather their descriptions) can be written as words over some alphabet.

Moreover, we can write configurations in such a way that the length of the corresponding words will be approximately the same as the amount of memory used by the algorithm (i.e., the number of cells on the tape used by a Turing machine, the number of number of bits of memory used by a RAM, etc.).

**Remark:** If we have an alphabet $\Sigma$ where $|\Sigma| = c$ then:

- The number of words of length $n$ is $c^n$, i.e., $2^{\Theta(n)}$.
- The number of words of length at most $n$ is

  $$\sum_{i=0}^{n} c^n = \frac{c^{n+1} - 1}{c - 1}$$

  i.e., also $2^{\Theta(n)}$. 
It is clear that during a computation of an algorithm there is no configuration repeated, since otherwise the computation would loop.

Therefore, if we know that the space complexity of an algorithm is $O(f(n))$, it means that the number of different configurations that are reachable during a computation is $2^{O(f(n))}$.

Since configurations do not repeat during a computation, also the time complexity of the algorithm is at most $2^{O(f(n))}$.

**Observation**

For every function $f : \mathbb{N} \rightarrow \mathbb{N}$ it holds that $S(f(n)) \subseteq T(2^f(n))$. 
The following results can be drawn from the previous discussion:

\[
\text{LOGSPACE} \subseteq \text{PTIME} \\
\text{PSPACE} \subseteq \text{EXPTIME}
\]
Summary:

\[
\text{LOGSPACE } \subseteq \text{PTIME } \subseteq \text{PSPACE } \subseteq \text{EXPTIME } \subseteq \text{EXPSPACE}
\]

- \( \text{PTIME } \subsetneq \text{EXPTIME} \)
- \( \text{LOGSPACE } \subsetneq \text{PSPACE} \)
An upper bound on a complexity of a problem means that the complexity of the problem is not greater than some specified complexity. Usually it is formulated so that the problem belongs to a particular complexity class.

Examples of propositions dealing with upper bounds on the complexity:

- The problem of reachability in a graph is in $\text{PTIME}$.  
- The problem of equivalence of two regular expressions is in $\text{EXPSPACE}$.  

If we want to find some upper bound on the complexity of a problem it is sufficient to show that there is an algorithm with a given complexity.
A **lower bound** on a complexity of a problem means that the complexity of the problem is at least as big as some specified complexity.

In general, proving of (nontrivial) lower bounds is more difficult than proving of upper bounds.

To derive a lower bound we must prove that *every* algorithm solving the given problem has the given complexity.
Problem “Sorting”

<table>
<thead>
<tr>
<th>Input:</th>
<th>Sequence of elements $a_1, a_2, \ldots, a_n$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>Elements $a_1, a_2, \ldots, a_n$ sorted from the smallest to the greatest.</td>
</tr>
</tbody>
</table>

It can be proven that every algorithm, that solves the problem “Sorting” and that has the property that the only operation applied on elements of a sorted sequence is a comparison (i.e., it does not examine the content of these elements), has the time complexity in the worst case $\Omega(n \log n)$ (i.e., for every such algorithm there exist constants $c > 0$ and $n \geq n_0$ such that for every $n \geq n_0$ there is an input of size $n$, for which the algorithm performs at least $cn \log n$ operations.)
Nondeterminism
Nondeterminism

Nondeterministic RAM:

- Its definition is very similar to that of a deterministic RAM.
- Moreover, it has an instruction $\text{NDJUMP } x \text{ OR } y$ that allows it to choose the next instruction from two possibilities.
- If at least one of computations of such a machine on a given input ends with the answer $\text{Yes}$, then the answer is $\text{Yes}$.
- If all computations end with the answer $\text{No}$ then the answer is $\text{No}$.

Nondeterministic versions of other computational models (such as nondeterministic Turing machines) are defined similarly.
The time required for a computation of a nondeterministic RAM (or other nondeterministic machine) on a given input is defined as the length of the longest computation on the input.
The time required for a computation of a nondeterministic RAM (or other nondeterministic machine) on a given input is defined as the length of the longest computation on the input.
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?

$k = 3$
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?

A nondeterministic algorithm works as follows:

1. It assigns nondeterministically to every node of $G$ one of $k$ colors.

2. It goes through all edges of $G$ and for each of them verifies that its endpoints are colored with different colors. If this is not the case, it halts with the answer No.

3. If it has verified for all edges that their endpoints are colored with different colors, it halts with the answer Yes.
Problem “Graph isomorphism”

Input: Undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.

Question: Are graphs $G_1$ and $G_2$ isomorphic?

Remark: Graphs $G_1$ and $G_2$ are isomorphic if there exists some bijection $f : V_1 \rightarrow V_2$ such that for every pair of nodes $u, v \in V_1$ is $(u, v) \in E_1$ iff $(f(u), f(v)) \in E_2$.

A nondeterministic algorithm works as follows:

1. It nondeterministically chooses values of the function $f$ for every $v \in V_1$.
2. It (deterministically) verifies that $f$ is a bijection and that the above mentioned condition is satisfied for all pairs of nodes.
3. If some of the conditions is violated, it halts with the answer No. Otherwise it halts with the answer Yes.
Nondeterminism can be viewed in two different ways:

1. When a machine should nondeterministically choose between several possibilities, it “guesses” which of these possibilities will lead to the answer \textbf{YES} (if there is such a possibility).

2. When a machine should choose between several possibilities, it splits itself into several copies, each corresponding to one of the possibilities. These copies continue in the computation in parallel. The answer is \textbf{YES} iff at least one of these copies halts with the answer \textbf{YES}. 
For decidability of problems, the nondeterministic algorithms are not more powerful than deterministic ones:
If a problem can be solved by a nondeterministic RAM or TM, it can be also solved by a deterministic RAM or TM that successively tries all possible computations of the nondeterministic machine on a given input.

Nondeterminism is useful primarily in the study of a complexity of problems.
Nondeterministic Complexity Classes

**Definition**

For a function $f : \mathbb{N} \rightarrow \mathbb{N}$ we define the **time complexity class** $\mathcal{NT}(f)$ as the set of all problems that are solved by nondeterministic RAMs with a time complexity in $O(f(n))$.

**Definition**

For a function $f : \mathbb{N} \rightarrow \mathbb{N}$ we define the **space complexity class** $\mathcal{NS}(f)$ as the set of all problems that are solved by nondeterministic RAMs with a space complexity in $O(f(n))$. 
Class NPTIME

Definition

\[ \text{NPTIME} = \bigcup_{k=0}^{\infty} \mathcal{N}^T(n^k) \]

- **NPTIME** (sometimes we write just **NP**) is the class of all problems, for which there exists a nondeterministic algorithm with polynomial time complexity.

- The class **NPTIME** contains those problems for which it is possible to verify in polynomial time that the answer is **Yes** if somebody, who wants to convince us that this is really the case, provides additional information.

- It is obvious that **PTIME \subseteq NPTIME**, since deterministic algorithms can be viewed as a special case of nondeterministic algorithms.
Complexity Classes

- Class **NPSPACE** can be defined similarly.
- Similarly to the deterministic case, it holds that **NPTIME \(\subseteq\) NPSPACE**.
- A nondeterministic RAM can be simulated by a deterministic RAM which tries all possible computations. Each of these possible computations uses at most a polynomial amount of memory and individual computations share no information so we can use the same memory for each of them. So we have **NPTIME \(\subseteq\) PSPACE**.
- It is also known that if some problem is solved by a nondeterministic Turing machine with a space complexity \(f(n)\) then it is also solved by a deterministic Turing machine with a space complexity \(O((f(n))^2)\). Based on this it is evident that **NPSPACE \(\subseteq\) PSPACE**.

\[
\text{PTIME} \subseteq \text{NPTIME} \subseteq \text{PSPACE} = \text{NPSPACE}
\]
NP-Complete Problems
Let us have two decision problems \( A \) and \( B \).

Problem \( A \) can be reduced to problem \( B \) if there is an algorithm \( P \) such that:

- It can get an arbitrary instance of problem \( A \) as an input.
- For an instance of a problem \( A \) obtained as an input (let us denote it as \( x \)) it produces an instance of a problem \( B \) as an output.
- It holds \( x \in A \iff P(x) \in B \)

i.e., the answer for the input \( x \) of problem \( A \) is \textbf{YES} iff the answer for the input \( P(x) \) of problem \( B \) is \textbf{YES}.

If, in addition, the algorithm \( P \) is polynomial, we say that problem \( A \) can be reduced in polynomial time to problem \( B \).
Polynomial Reduction between Problems

Inputs of problem $A$

- Yes
- No

Inputs of problem $B$

- Yes
- No
Polynomial Reduction between Problems

Inputs of problem $A$ \hspace{2cm} Inputs of problem $B$

\begin{tabular}{c c c}
Yes & Yes & Yes \\
No & & NO \\
\end{tabular}

$P$
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 $\text{PTIME}$. 
Definition of problem SAT:

**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 $[x_1]_\nu = 1$, $[x_2]_\nu = 0$, $[x_3]_\nu = 1$, it holds that $[\varphi_1]_\nu = 1$.

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 $[\varphi_2]_\nu = 0$. 

3-SAT is a variant of the SAT problem where the possible inputs are restricted to formulas of a certain special form:

**3-SAT**

**Input:** Formula \( \varphi \) is a conjunctive normal form where every clause contains exactly 3 literals.

**Question:** Is \( \varphi \) satisfiable?
Some notions:

- A **literal** is a formula of the form $x$ or $\neg x$ where $x$ is a boolean variable.

- A **clause** is a disjunction of literals.
  
  Examples: $x_1 \lor \neg x_2 \quad \neg x_5 \lor x_8 \lor \neg x_{15} \lor \neg x_{23} \quad x_6$

- 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:**

$$(x_1 \lor \neg x_2 \lor x_4) \land (\neg x_1 \lor x_3 \lor x_3) \land (\neg x_1 \lor \neg x_3 \lor \neg x_4) \land (x_2 \lor \neg x_3 \lor x_4)$$
Problem 3-SAT

The following formula is satisfiable:

\[(x_1 \lor \neg x_2 \lor x_4) \land (\neg x_1 \lor x_3 \lor x_3) \land (\neg x_1 \lor \neg x_3 \lor \neg x_4) \land (x_2 \lor \neg x_3 \lor x_4)\]

For example for valuation \(\nu\) where

\[
\begin{align*}
[x_1]_{\nu} &= 0 \\
[x_2]_{\nu} &= 1 \\
[x_3]_{\nu} &= 0 \\
[x_4]_{\nu} &= 1
\end{align*}
\]

is \([\varphi_1]_{\nu} = 1\).

On the other hand, the following formula is not satisfiable:

\[(x_1 \lor x_1 \lor x_1) \land (\neg x_1 \lor \neg x_1 \lor \neg x_1)\]
Independent Set (IS) Problem

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.
An example of an instance where the answer is **Yes**:

\[ k = 4 \]

An example of an instance where the answer is **No**:

\[ k = 5 \]
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 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 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

\((x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4) \land (x_1 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor x_2 \lor x_4)\)

\(\nu(x_1) = 1\)
\(\nu(x_2) = 1\)
\(\nu(x_3) = 0\)
\(\nu(x_4) = 1\)

If the formula \(\varphi\) is satisfiable then there is a valuation \(\nu\) where every clause contains at least one literal with value 1.
A Reduction from 3-SAT to IS

\((x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4) \land (x_1 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor x_2 \lor x_4)\)

\[\begin{align*}
\nu(x_1) &= 1 \\
\nu(x_2) &= 1 \\
\nu(x_3) &= 0 \\
\nu(x_4) &= 1
\end{align*}\]

We select one literal that has a value 1 in the valuation \(\nu\), and we put the corresponding node into the independent set.
A Reduction from 3-SAT to IS

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4) \land (x_1 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor x_2 \lor x_4)\]

\[
\nu(x_1) = 1 \\
\nu(x_2) = 1 \\
\nu(x_3) = 0 \\
\nu(x_4) = 1 \\
\]

We can easily verify that the selected nodes form an independent set.
A Reduction from 3-SAT to IS

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 $\neg x_i$ could not be selected together. (Exactly of them has the value 1 in the given valuation $\nu$.)
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 $\phi$ in time $O(n^2)$, where $n$ is the size of formula $\phi$.

We have also seen that there is an independent set of size $k$ in the constructed graph $G$ iff the formula $\phi$ is satisfiable.

The described algorithm shows that 3-SAT can be reduced in polynomial time to IS.
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.
NP-Complete Problems

Let us consider now the class $\text{NPTIME}$ and a problem $P$. 
A problem $P$ is **NP-hard** if every problem from 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 \( \text{NPTIME} \):

- 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 \( \text{NPTIME} \) there would be a polynomial-time (deterministic) algorithm.
On the other hand, if there is at least one problem from \textit{NPTIME} for which a polynomial-time algorithm does not exist, then it means that for none of \textit{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.
NP-Complete Problems

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 obvious that 3-SAT and IS are in 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
- IS
- 3-SAT
- HC
- HK
- TSP
- VC
- CLIQUE

Z. Sawa (TU Ostrava)  Introd. to Theoretical Computer Science  February 6, 2014  364 / 401
3-CG – Graph Coloring with 3 Colors

3-CG - graph coloring with 3 colors

Input: An undirected graph $G$

Question: Is it possible to color nodes of the graph $G$ using 3 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:
3-CG – Graph Coloring with 3 Colors

3-CG - graph coloring with 3 colors

Input: An undirected graph $G$

Question: Is it possible to color nodes of the graph $G$ using 3 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:

Answer: Yes
3-CG – Graph Coloring with 3 Colors

3-CG - graph coloring with 3 colors

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

**Question:** Is it possible to color nodes of the graph $G$ using 3 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:**

![Example Graph](image)
3-CG – Graph Coloring with 3 Colors

Input: An undirected graph $G$

Question: Is it possible to color nodes of the graph $G$ using 3 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:

![Graph Coloring Example](image)

Answer: No
**IS – Independent Set**

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

**Question:** Is there an independent set of size $k$ in graph $G$ (i.e., a set of $k$ nodes where no pair of nodes from this set is connected by an edge)?

**Example:** $k = 5$
Input: An undirected graph $G$ and a natural number $k$.

Question: Is there an independent set of size $k$ in graph $G$ (i.e., a set of $k$ nodes where no pair of nodes from this set is connected by an edge)?

Example: $k = 5$

Answer: $\text{YES}$
**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
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)
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 with arrows showing a Hamiltonian cycle through each node exactly once.](diagram)

---

Z. Sawa (TU Ostrava)  Introd. to Theoretical Computer Science  February 6, 2014  369 / 401
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:**

![Example Diagram]

**Answer:** Yes
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
```
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:
## 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:

![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 example](image)

**Answer:** 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 sum is a special case of the Knapsack problem can be seen from the following (almost trivial) reduction:

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 numbers in an input encoded in **binary**, i.e., the size of an input is proportional to the sum of lengths of binary representations of numbers in the input, then the problem SUBSET-SUM is **NP**-hard. (There is a polynomial time reduction from 3-SAT.)
It is not difficult to see that SUBSET-SUM and Knapsack problem (its decision variant) belong to the class \textbf{NPTIME}:

- A nondeterministic algorithm at first nondeterministically chooses a subset of elements of the sequence in the input and then (deterministically) verifies if it satisfies the given condition (resp. conditions).

It is obvious that this verification can be done in time polynomial with respect to the size of the instance.

So the problems SUBSET-SUM and Knapsack problem are \textbf{NP}-complete.
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:

$$
A = \begin{pmatrix}
3 & -2 & 5 \\
1 & 0 & 1 \\
2 & 1 & 0
\end{pmatrix} \\
b = \begin{pmatrix}
8 \\
-3 \\
5
\end{pmatrix}
$$

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 **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.
As already mentioned, the question if $\text{PTIME} = \text{NPTIME}$ is a longstanding open problem.

In general, it is supposed that most likely $\text{PTIME} \neq \text{NPTIME}$ but nobody has proved it yet.

**Remark:** For example it can be shown that $\text{PTIME} \neq \text{NPTIME}$ is a necessary (but not sufficient) condition for an existence of ciphers that are not easily breakable.

If somebody did find a polynomial algorithm for at least one $\text{NP}$-complete problem, we would have immediately obtained algorithms for fast breaking of all ciphers that are currently used.
Undecidable Problems
A problem that is not algorithmically solvable is **algorithmically unsolvable**.

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

**Example**: The following problem called the **Halting problem** is undecidable:

### 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?
Alternatively, we could define the problem as follows:

**Halting problem**

**Input:** A source code of a C 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

$\text{boolean } H(\text{String code, String input})$

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.
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 `String`.
- It calls the subroutine $H(x, x)$.
- If subroutine $H$ returns `true`, program $D$ jumps into an infinite loop

```
loop: goto loop
```

In case that $H$ returns `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 $\text{true}$ and $D$ jumps into the infinite loop. A contradiction!
- If $D$ does not halt then $H(D, D)$ returns $\text{false}$ 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.
Other Undecidable Problems

We have already seen an 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>
Reduction between Problems

If we have already proved a (decision) problem to be undecidable, we can prove undecidability of other problems by reductions.

Let us say that $A$ and $B$ are decision problems. A **reduction** of problem $A$ to problem $B$ is an algorithm $P$ such that:

- It gets as an input an instance of problem $A$ (denoted $x$).
- It produces an instance of problem $B$ as an output (denoted $P(x)$).
- It holds for every instance $x$ of problem $A$ that the answer for $x$ in problem $A$ is $\text{YES}$ iff the answer for $P(x)$ in problem $B$ is $\text{YES}$. 
Reduction between Problems

Let us say there is some reduction $P$ from problem $A$ to problem $B$.

If problem $B$ is decidable then problem $A$ is also decidable.

Solution of problem $A$ for an input $x$:
- Call $P$ with $x$ as an input, it returns a value $P(x)$.
- Call the algorithm solving problem $B$ with input $P(x)$.
- Write the returned value to the output as the result.

It is obvious that if $A$ is undecidable then $B$ 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?
- ...

---

*Z. Sawa (TU Ostrava)*

*Introd. to Theoretical Computer Science*
Other Undecidable Problems

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

An input is a set of types of cards, such as:

```
<table>
<thead>
<tr>
<th>abb</th>
<th>a</th>
<th>bab</th>
<th>baba</th>
<th>aba</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbab</td>
<td>aa</td>
<td>ab</td>
<td>aa</td>
<td>a</td>
</tr>
</tbody>
</table>
```

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.

```
<table>
<thead>
<tr>
<th>a</th>
<th>abb</th>
<th>abb</th>
<th>baba</th>
<th>abb</th>
<th>aba</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa</td>
<td>bbab</td>
<td>bbab</td>
<td>aa</td>
<td>bbab</td>
<td>a</td>
</tr>
</tbody>
</table>
```

In the upper and in the lower row we obtained the word

```
aabbabbabaabbaba.
```
Other Undecidable Problems

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 FOL where $+$, $\times$, and integer constants can be used as function symbols and $=$ and $<$ as predicate symbols.

**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:

$$\forall x \exists y \forall z ((x \times y = z) \land (y + 5 = x))$$

**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.
If the function symbol $*$ 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

$$\exists x \exists y \exists z (5 * x * x * y + (-8) * y * z + 3 * z * z + (-15) = 0)$$

holds in the domain of natural numbers.
Also the following problem is algorithmically undecidable:

**Problem**

**Input:** A closed formula $\varphi$ of FOL.
**Question:** Is $\models \varphi$?

**Remark:** Notation $\models \varphi$ denotes that formula $\varphi$ is logically valid, i.e., it is true in all interpretations.
A problem is **semidecidable** if there is an algorithm such that:

- If it gets as an input an instance for which the answer is **Yes**, then it halts after some finite number of steps and writes "**YES**" on the output.

- If it gets as an input an instance for which the answer is **No**, then it either halts and writes "**NO**" on the input, or does not halt and runs forever.

It is obvious that for example HP (Halting Problem) is semidecidable.

Some problems are not even semidecidable.