Automata Theory and Formal Languages

Summer Term 2026

16th Lecture

Turing Machines

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

Turing Machine

Type 0 and Type 1 Languages

Type 0

No restrictions for productions P

Type 0 and Type 1 Languages

Type 0

No restrictions for productions P

Type 1

$\forall(w_1 \to w_2) \in P : |w_1| \leq |w_2|$

Thus no shortening rules are allowed. These rules are also called *monotonic*.

Type 0 and Type 1 Languages

Type 0

No restrictions for productions P

Type 1

$\forall(w_1 \to w_2) \in P : |w_1| \leq |w_2|$

Thus no shortening rules are allowed. These rules are also called *monotonic*.

Remark

For type 0 and type 1 grammars it is also allowed that terminals are on the left hand side. The only restriction was, that at least one variable is on the left hand side.

Type 1 Language: Example

As discussed before, the language

$$L = \{a^n b^n c^n \mid n > 1\}$$

is not context-free. It thus cannot be accepted by a PDA.

Type 1 Language: Example

As discussed before, the language

$$L = \{a^n b^n c^n \mid n > 1\}$$

is not context-free. It thus cannot be accepted by a PDA.

Question: Is L context-sensitive?

Type 1 Language: Example

As discussed before, the language

$$L = \{a^n b^n c^n \mid n > 1\}$$

is not context-free. It thus cannot be accepted by a PDA.

Question: Is L context-sensitive?

Answer: yes!

Type 1 Language: Example

Grammar:

$$
\begin{aligned}
S &\to aSBC \mid aBC \\
CB &\to BC \\
aB &\to ab \\
bB &\to bb \\
bC &\to bc \\
cC &\to cc
\end{aligned}
$$

Type 1 Language: Example

Grammar:

$$
\begin{aligned}
S &\to aSBC \mid aBC \\
CB &\to BC \\
aB &\to ab \\
bB &\to bb \\
bC &\to bc \\
cC &\to cc
\end{aligned}
$$

Example derivation for $aaabbbccc$**:**

$$
\begin{aligned}
S &\Rightarrow aSBC \Rightarrow aaSBCBC \Rightarrow aaaBCBCBC \\
&\Rightarrow aaaBBCCBC \Rightarrow aaaBBCBCC \\
&\Rightarrow aaaBBBCCC \\
&\Rightarrow aaabBBCCC \\
&\Rightarrow aaabbBCCC \\
&\Rightarrow aaabbbCCC \\
&\Rightarrow aaabbbcCC \\
&\Rightarrow aaabbbccC \\
&\Rightarrow aaabbbccc
\end{aligned}
$$

Type 1 Language: Example

Remarks

- Grammar has several context-sensitive rules
- These are necessary to prevent that other strings can be derived. For instance rules like $B \to b$ would generate other strings.

Automaton Model

Automaton model for type 0/1 languages?

Automaton Model

Automaton model for type 0/1 languages?

Turing machine!

Alan M. Turing

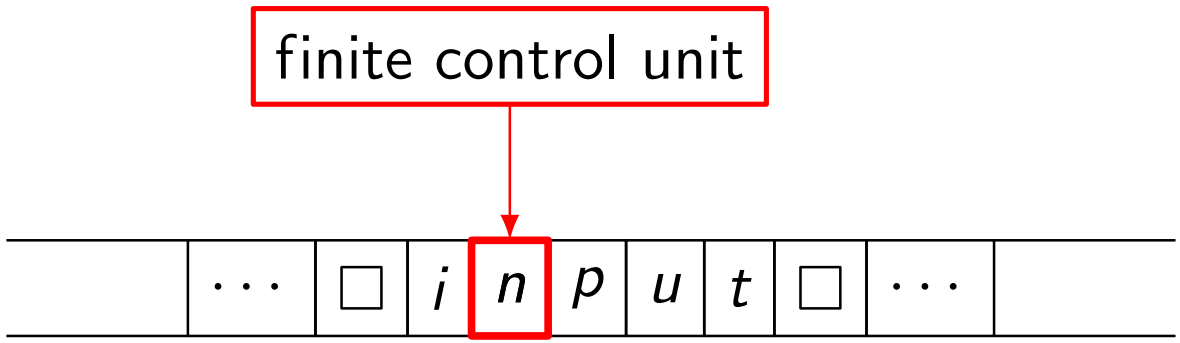
- 1912 – 1954
- English mathematician
- Inventor of the Turing Machine (1937) proving the foundation for the definition of computability
- Turing test more machine intelligence
- Decryption of the Enigma (second world war)

Turing Machine

- Automaton that is more powerful than a PDA.
- It defines what we think as intuitively computable. No programming language / machine model known until now is more powerful.
- A Turing machine consists of
  - an infinite tape
  - a read/write head
  - a control unit

Turing Machine

- Automaton that is more powerful than a PDA.
- It defines what we think as intuitively computable. No programming language / machine model known until now is more powerful.
- A Turing machine consists of
  - an infinite tape
  - a read/write head
  - a control unit



Turing Machine

Remark

$\square$ is the blank sign. All fields of the tape that have not yet been visited will have the $\square$ field.

Turing Machine

Definition

A *Turing Machine* (TM) M is a 7 tuple

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E) \quad \text{where}$$

Turing Machine

Definition

A *Turing Machine* (TM) M is a 7 tuple

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E) \quad \text{where}$$

- Z: Finite set of *states*

Turing Machine

Definition

A *Turing Machine* (TM) M is a 7 tuple

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E) \quad \text{where}$$

- Z: Finite set of *states*
- Σ: Input alphabet

Turing Machine

Definition

A *Turing Machine* (TM) M is a 7 tuple

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E) \quad \text{where}$$

- Z: Finite set of *states*
- Σ: Input alphabet
- $\Gamma \supset \Sigma$: Tape alphabet

Turing Machine

Definition

A *Turing Machine* (TM) M is a 7 tuple

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E) \quad \text{where}$$

- Z: Finite set of *states*
- Σ: Input alphabet
- $\Gamma \supset \Sigma$: Tape alphabet
- $z_0 \in Z$: Initial state

Turing Machine

Definition

A *Turing Machine* (TM) M is a 7 tuple

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E) \quad \text{where}$$

- Z: Finite set of *states*
- Σ: Input alphabet
- $\Gamma \supset \Sigma$: Tape alphabet
- $z_0 \in Z$: Initial state
- $\square \in \Gamma \setminus \Sigma$: Blank symbol

Turing Machine

Definition

A *Turing Machine* (TM) M is a 7 tuple

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E) \quad \text{where}$$

- Z: Finite set of *states*
- Σ: Input alphabet
- $\Gamma \supset \Sigma$: Tape alphabet
- $z_0 \in Z$: Initial state
- $\square \in \Gamma \setminus \Sigma$: Blank symbol
- $E \subseteq Z$: Accepting states

Turing Machine

Definition

A *Turing Machine* (TM) M is a 7 tuple

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E) \quad \text{where}$$

- Z: Finite set of *states*
- Σ: Input alphabet
- $\Gamma \supset \Sigma$: Tape alphabet
- $z_0 \in Z$: Initial state
- $\square \in \Gamma \setminus \Sigma$: Blank symbol
- $E \subseteq Z$: Accepting states
- δ: $Z \times \Gamma \to Z \times \Gamma \times \{L, R, N\}$ (deterministic)
  δ: $Z \times \Gamma \to \mathcal{P}(Z \times \Gamma \times \{L, R, N\})$ (non-deterministic)

Turing Machine

$\delta(z, a) = (z', b, x)$ or $\delta(z, a) \ni (z', b, x)$ means:

- M is in state z and reads a
- it writes b
- switches into state z'
- read/write head moves to the right, to the left, or it remains

Configuration

A *configuration* is a string $k \in \Gamma^* Z \Gamma^*$ and describes a snapshot of a TM.

$$k = \alpha z \beta$$

means

- TM has $\alpha\beta$ on its tape (blank symbols to the left and right)
- TM is in state z
- Head position is at the first letter of β
- In case that TM is on a blank sign, one has to extend α / β.

Configuration

Based on the set of all possible configurations (for a given deterministic TM) we define the relation $\vdash$ as:

$$a_1 \dots a_m z b_1 \dots b_n \vdash \begin{cases} a_1 \dots a_m z' c b_2 \dots b_n & \text{if} \quad \delta(z, b_1) = (z', c, N), m \geq 0, n \geq 1 \\ a_1 \dots a_m c z' b_2 \dots b_n & \text{if} \quad \delta(z, b_1) = (z', c, R), m \geq 0, n \geq 2 \\ a_1 \dots a_{m-1} z' a_m c b_2 \dots b_n & \text{if} \quad \delta(z, b_1) = (z', c, L), m \geq 1, n \geq 1 \end{cases}$$

Special case if $n = 1$ and the TM turns to the right:

$$a_1 \dots a_m z b_1 \vdash a_1 \dots a_m c z' \square \quad \text{if} \quad \delta(z, b_1) = (z', c, R)$$

Special case if $m = 0$ and the TM turns to the left:

$$z b_1 \dots b_n \vdash z' \square c b_2 \dots b_n \quad \text{if} \quad \delta(z, b_1) = (z', c, L)$$

For a noneterministic TM $\delta(z, b_1) = (z', c,)$ needs to be replaced by $\delta(z, b_1) \ni (z', c,)$ in all definitions.

Accepted Language

Now we can define the language accepted by a TM M as

$$L(M) = \{x \in \Sigma^* \mid z_0 x \vdash^* \alpha z \beta \text{ where } \alpha, \beta \in \Gamma^* \ \wedge \ z \in E\}$$

Turing Machine

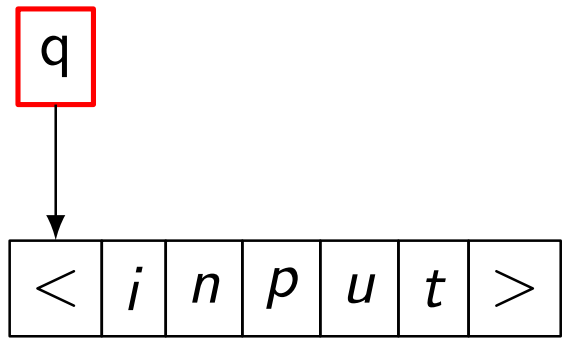
Theorem

The type 0 language are equivalent to the set of languages that can be accepted by a TM. Both non-deterministic and deterministic TMs are equivalent.

(without proof)

Linear Bounded Automaton (LBA)

- An LBA is a TM where the tape is bounded to the input.
- The read/write head is not allowed to modify symbols outside the input
- For convenience one uses two different blank symbols. The field to the left of the input is $<$. The symbol to the right is $>$.

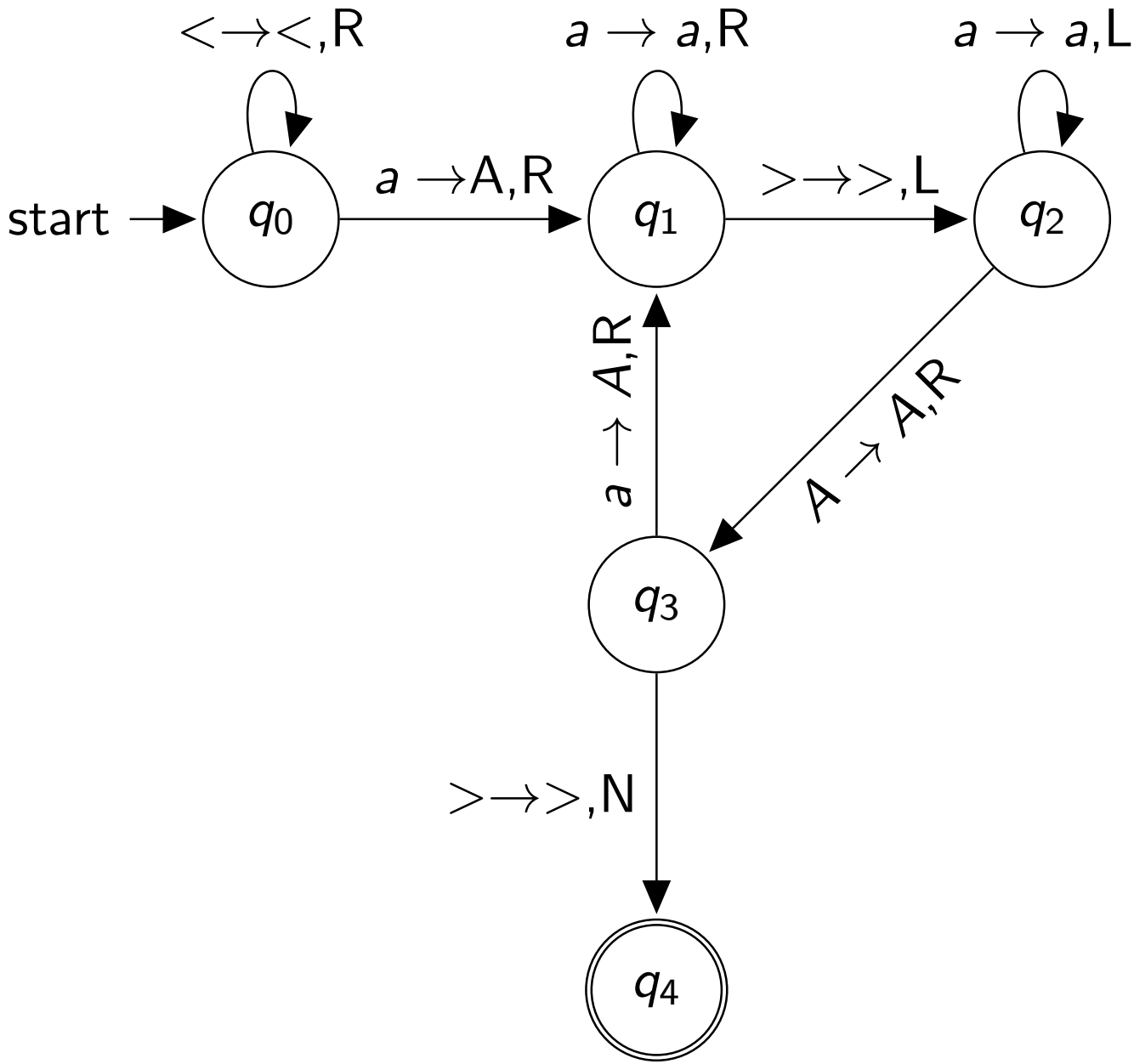


Example

We consider a linear bounded TM that gets as input N a's (a^N) and should accept this language in the following way:

- The TM should replace the farthest left small a by a capital A.
- It should then run to the right border without changing the tape.
- It then runs to the left to the farthest left small a and again replaces it by a capital A.
- The TM should proceed until all a's are capitalized. It should then go into the accepting state.

Example



Linear Bounded Automaton

Theorem (Kuroda)

The languages that can be accepted by a linear bounded TM (LBA) are exactly the type 1 languages.

(without proof)

Linear Bounded Automaton

Theorem (Kuroda)

The languages that can be accepted by a linear bounded TM (LBA) are exactly the type 1 languages.

(without proof)

Remark

If one looks at the derivation

$$S \Rightarrow \cdots \Rightarrow x$$

for a type 1 grammar one can see that the intermediate strings x_j are always shorter or as long as x. When an LBA simulates a type 1 grammar it thus does not need to leave the input string. This property is used in th proof of the Kuroda theorem (see Schöning for details).