Automata Theory and Formal Languages

Summer Term 2026

13th Lecture

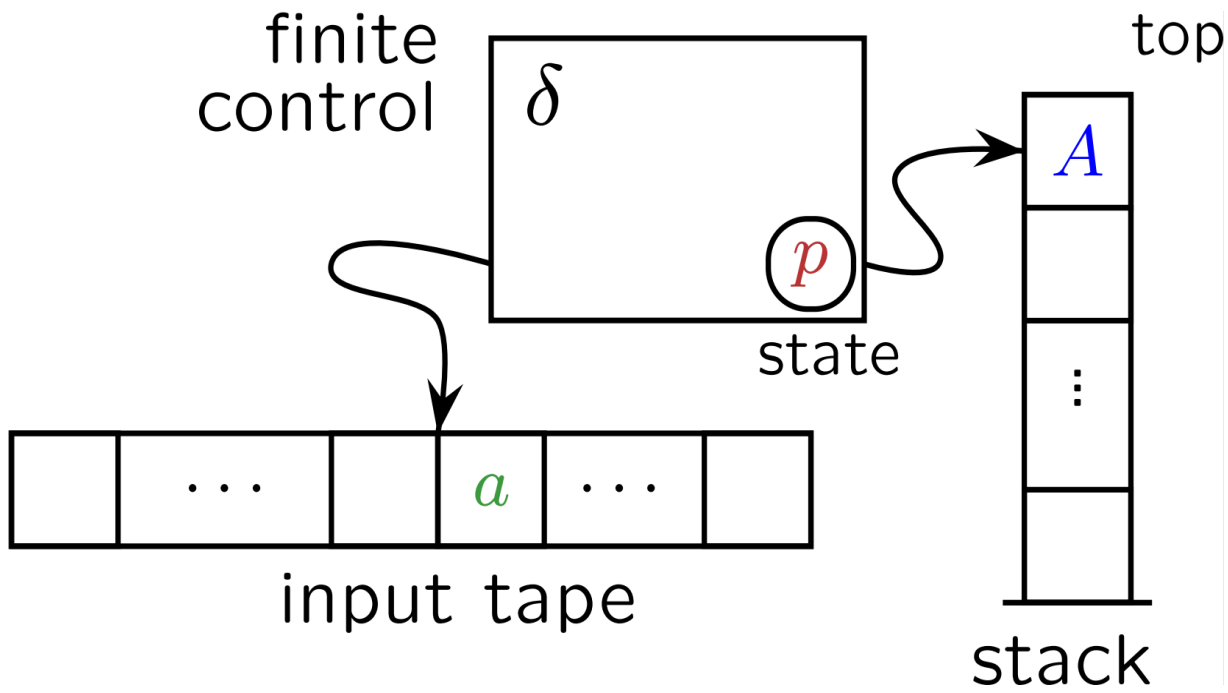
Pushdown Automaton

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

Pushdown Automaton

Pushdown Automaton

- A pushdown automaton is an extension of the NFA and introduces an additional stack.
- It works according to the LIFO (last in first out) principle.



Pushdown Automaton

Definition

A (non-deterministic) *Pushdown Automaton* M is a 6 tuple

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

Pushdown Automaton

Definition

A (non-deterministic) *Pushdown Automaton* M is a 6 tuple

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

- Z: Finite set of *states*

Pushdown Automaton

Definition

A (non-deterministic) *Pushdown Automaton* M is a 6 tuple

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

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

Pushdown Automaton

Definition

A (non-deterministic) *Pushdown Automaton* M is a 6 tuple

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

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

Pushdown Automaton

Definition

A (non-deterministic) *Pushdown Automaton* M is a 6 tuple

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

- Z: Finite set of *states*
- Σ: Input alphabet
- Γ: Stack alphabet
- $z_0 \in Z$: Initial state

Pushdown Automaton

Definition

A (non-deterministic) *Pushdown Automaton* M is a 6 tuple

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

- Z: Finite set of *states*
- Σ: Input alphabet
- Γ: Stack alphabet
- $z_0 \in Z$: Initial state
- $\# \in \Gamma$: Initial stack symbol

Pushdown Automaton

Definition

A (non-deterministic) *Pushdown Automaton* M is a 6 tuple

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

- Z: Finite set of *states*
- Σ: Input alphabet
- Γ: Stack alphabet
- $z_0 \in Z$: Initial state
- $\# \in \Gamma$: Initial stack symbol
- δ: $Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}_e(Z \times \Gamma^*)$ State transition function

  Here, $\mathcal{P}_e(Z \times \Gamma^*)$ is the set of all *finite* subsets of $Z \times \Gamma^*$.

Pushdown Automaton

Remarks

- A PDA allows for ε transitions

Pushdown Automaton

Remarks

- A PDA allows for ε transitions
- In each step the top stack symbol is read and it is
  - removed (POP),
  - replaced, or
  - extended (PUSH)

Pushdown Automaton

Remarks

- A PDA allows for ε transitions
- In each step the top stack symbol is read and it is
  - removed (POP),
  - replaced, or
  - extended (PUSH)
- Intuitively $\delta(z, a, A) \ni (z', B_1, \ldots, B_k)$ means that M is in state z, reads an a while A is the top stack symbol. The machine then transitions into state z' while the A on the stack is replaced by the symbols $B_1, \ldots, B_k$. Here, B_1 is the uppermost stack symbol while B_k is at the same stack position as A.

PDA – Acceptance

- As we have defined the PDA, it has no accepting states. It instead accepts with an *empty stack*. One may alternatively define the PDA by the 7 tuple

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \#, E)$$

where $E \subset Z$ are the accepting states.

PDA – Acceptance

- As we have defined the PDA, it has no accepting states. It instead accepts with an *empty stack*. One may alternatively define the PDA by the 7 tuple

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \#, E)$$

  where $E \subset Z$ are the accepting states.

- Any PDA that accepts with an empty stack can be converted into a PDA that accepts with accepting states and vice versa.

PDA – Configuration

A *configuration* k of a PDA is a tuple

$$k = (k_1, k_2, k_3) \in Z \times \Sigma^* \times \Gamma^*.$$

It is a snapshot of the PDA where

- $k_1 \in Z$ is the current state,
- $k_2 \in \Sigma^*$ is the remaining part of the input tape,
- $k_3 \in \Gamma^*$ is the current stack content.

PDA – Configuration

Based on the configuration we can define the relation $\vdash$ as

$$(z, a_1 \dots a_n, A_1 \dots A_m) \vdash (z', a_2 \dots a_n, B_1 \dots B_k A_2 \dots A_m) \quad \text{if}$$

$$\delta(z, a_1, A_1) \ni (z', B_1 \dots B_k)$$

and

$$(z, a_1 \dots a_n, A_1 \dots A_m) \vdash (z', a_1 \dots a_n, B_1 \dots B_k A_2 \dots A_m) \quad \text{if}$$

$$\delta(z, \varepsilon, A_1) \ni (z', B_1 \dots B_k)$$

PDA – Acceptance

Using the $\vdash$ relation, the language accepted by a PDA can be defined to be

$$L(M) = \{x \in \Sigma^* \mid (z_0, x, \#) \vdash^* (z, \varepsilon, \varepsilon) \text{ for a } z \in Z\}.$$

For a PDA with accepting states the accepted language is defines as

$$L(M) = \{x \in \Sigma^* \mid (z_0, x, \#) \vdash^* (z, \varepsilon, \gamma) \text{ for a } z \in E \text{ and } \gamma \in \Gamma^*\}.$$

Here, γ is the part that remains on the stack upon acceptance.

PDA – Notation

Instead of

$$\delta(z, a, A) \ni (z', \gamma)$$

one can write the short notation

$$zaA \to z'\gamma.$$

We will usually use that notation.

Example

We consider a PDA for the language

$$L = \{a_1 a_2 \dots a_n \$ a_n \dots a_2 a_1 \mid a_i \in \{a, b\}\}$$

Example

We consider a PDA for the language

$$L = \{a_1 a_2 \dots a_n \$ a_n \dots a_2 a_1 \mid a_i \in \{a, b\}\}$$

It is given by $M = (\{z_0, z_1\}, \{a, b, \$\}, \{\#, A, B\}, \delta, z_0, \#)$ with productions

$$z_0 a \# \xrightarrow{1} z_0 A \#, \quad z_0 a A \xrightarrow{2} z_0 A A, \quad z_0 a B \xrightarrow{3} z_0 A B$$

$$z_0 b \# \xrightarrow{4} z_0 B \#, \quad z_0 b A \xrightarrow{5} z_0 B A, \quad z_0 b B \xrightarrow{6} z_0 B B$$

$$z_0 \$ \# \xrightarrow{7} z_1 \#, \quad z_0 \$ A \xrightarrow{8} z_1 A, \quad z_0 \$ B \xrightarrow{9} z_1 B$$

$$z_1 a A \xrightarrow{10} z_1 \varepsilon, \quad z_1 b B \xrightarrow{11} z_1 \varepsilon, \quad z_1 \varepsilon \# \xrightarrow{12} z_1 \varepsilon$$

Here, we have enumerated the productions to later see, which is applied.

Example

Remarks

- The first 6 rules describe, how the first half of the string is read and how successively the stack is filled. For any a we put an A on the stack and for any b a B is put on the stack.

Example

Remarks

- The first 6 rules describe, how the first half of the string is read and how successively the stack is filled. For any a we put an A on the stack and for any b a B is put on the stack.
- Rule 7 – 9 describe how the automaton passes the \$ sign and then switched into state z_1. It then knows that it reads the second half of the string.

Example

Remarks

- The first 6 rules describe, how the first half of the string is read and how successively the stack is filled. For any a we put an A on the stack and for any b a B is put on the stack.
- Rule 7 – 9 describe how the automaton passes the \$ sign and then switched into state z_1. It then knows that it reads the second half of the string.
- Rules 10 – 12 are responsible of reducing the stack. Here, always the currently read symbol is compared to the stack symbol and the automaton can only proceed if they "match" (if we ignore capitalization).

Example

We now can show that the automaton accepts

$$ba\$ab$$

since

Example

We now can show that the automaton accepts

$$ba\$ab$$

since

$$(z_0, ba\$ab, \#) \overset{4}{\vdash} (z_0, a\$ab, B\#) \overset{3}{\vdash} (z_0, \$ab, AB\#)$$
$$\overset{8}{\vdash} (z_1, ab, AB\#) \overset{10}{\vdash} (z_1, b, B\#) \overset{11}{\vdash} (z_1, \varepsilon, \#) \overset{12}{\vdash} (z_1, \varepsilon, \varepsilon)$$

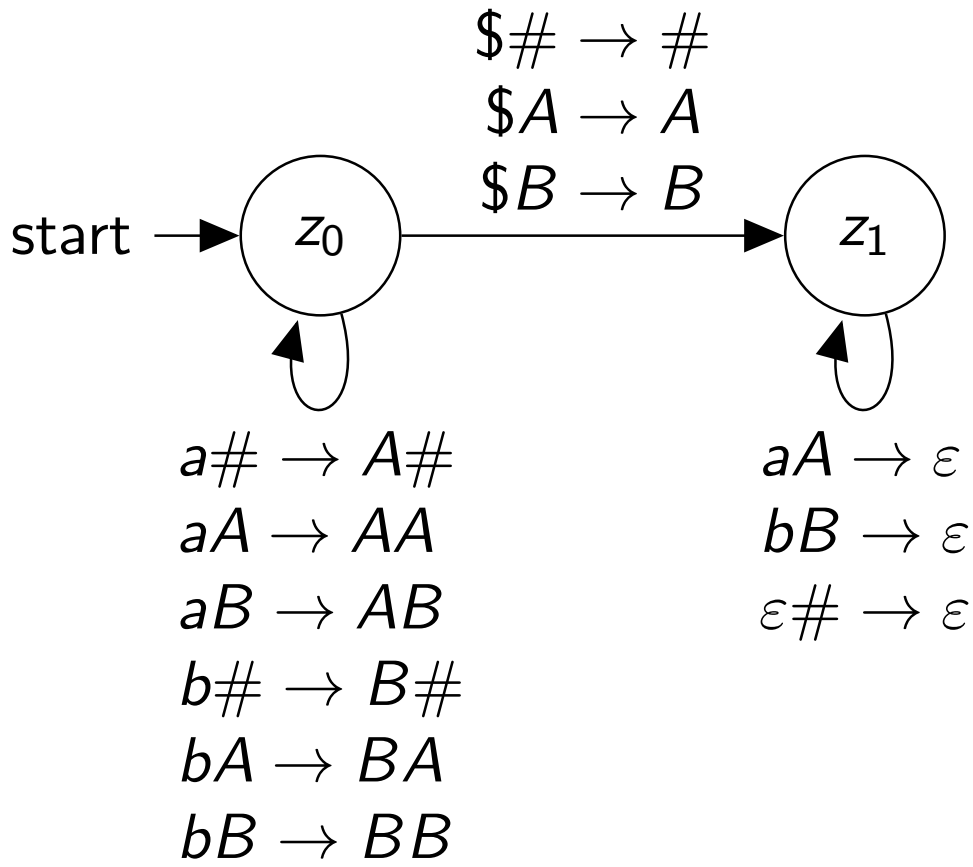
Here, we have put the number of the rule that has been applied over the $\vdash$ sign.

Pushdown Automaton

The δ function of a PDA can be expressed as a state diagram

Pushdown Automaton

The δ function of a PDA can be expressed as a state diagram



Note that the state has been removed from both sides of the edge labels.

Deterministic PDA

Remark

The PDA for the palindrome language discussed before was *deterministic*. In each step there was just one possible δ function that could be applied.

Deterministic PDA

Remark

The PDA for the palindrome language discussed before was *deterministic*. In each step there was just one possible δ function that could be applied.

If one replaces the \$ rules by

$$z_0 aA \to z_1 \varepsilon \quad z_0 bB \to z_1 \varepsilon \quad z_0 \varepsilon \# \to z_1 \varepsilon$$

one obtains a non-deterministic PDA M' for the language

$$L(M') = \{a_1 a_2 \dots a_n a_n \dots a_2 a_1 \mid a_i \in \{a, b\}\}$$

Deterministic PDA

Remark

The PDA for the palindrome language discussed before was *deterministic*. In each step there was just one possible δ function that could be applied.

If one replaces the \$ rules by

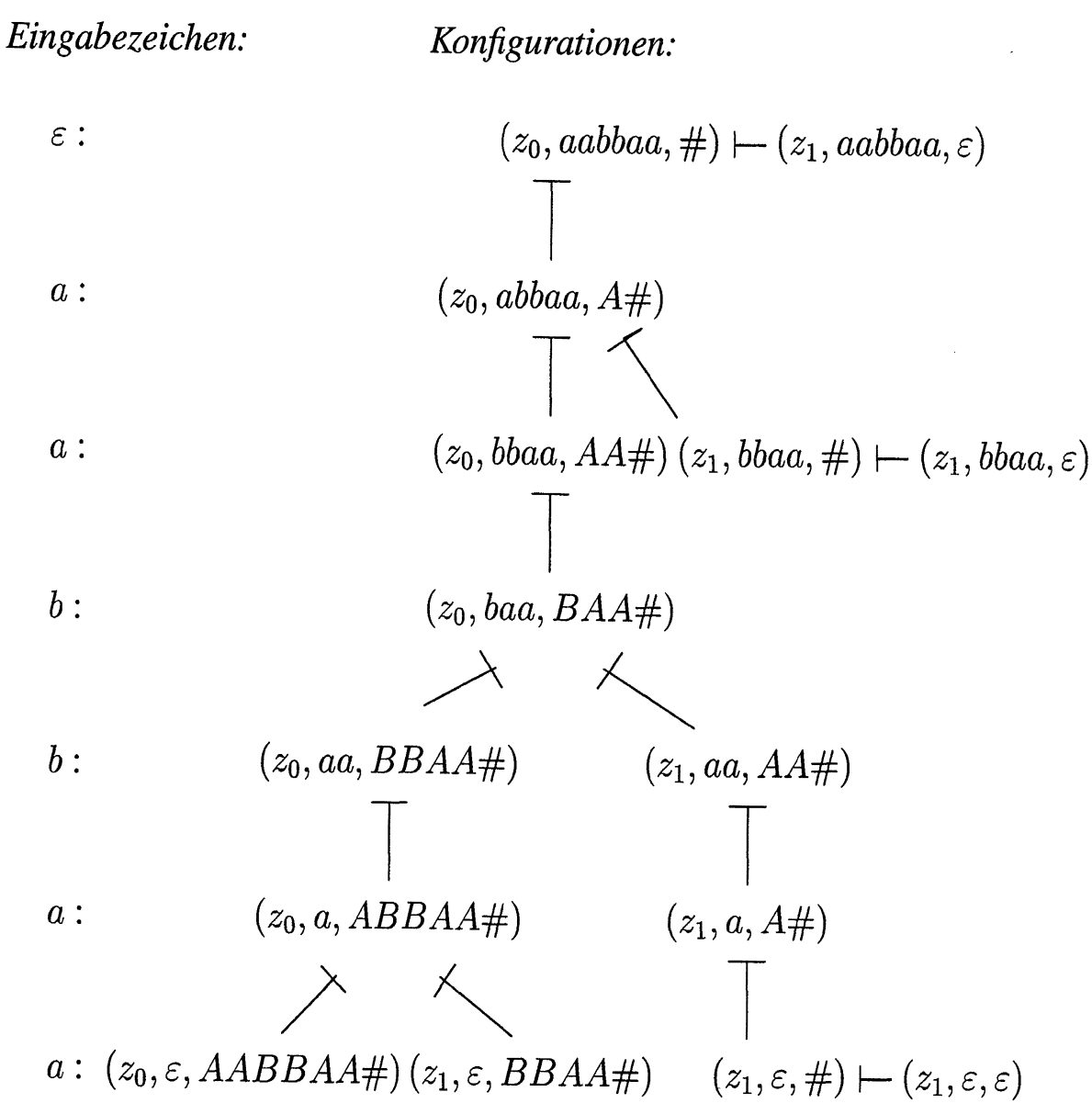
$$z_0 a A \to z_1 \varepsilon \quad z_0 b B \to z_1 \varepsilon \quad z_0 \varepsilon \# \to z_1 \varepsilon$$

one obtains a non-deterministic PDA M' for the language

$$L(M') = \{a_1 a_2 \dots a_n a_n \dots a_2 a_1 \mid a_i \in \{a, b\}\}$$

When reading the string *aabbaa* the following configurations are possible:

Deterministic PDA



Picture taken from Schöning.

Deterministic PDA

One can see that the non-determinism is required for *guessing* the middle of the string. One can show that the language M' cannot be accepted by a deterministic PDA.