Automata Theory and Formal Languages

Summer Term 2026

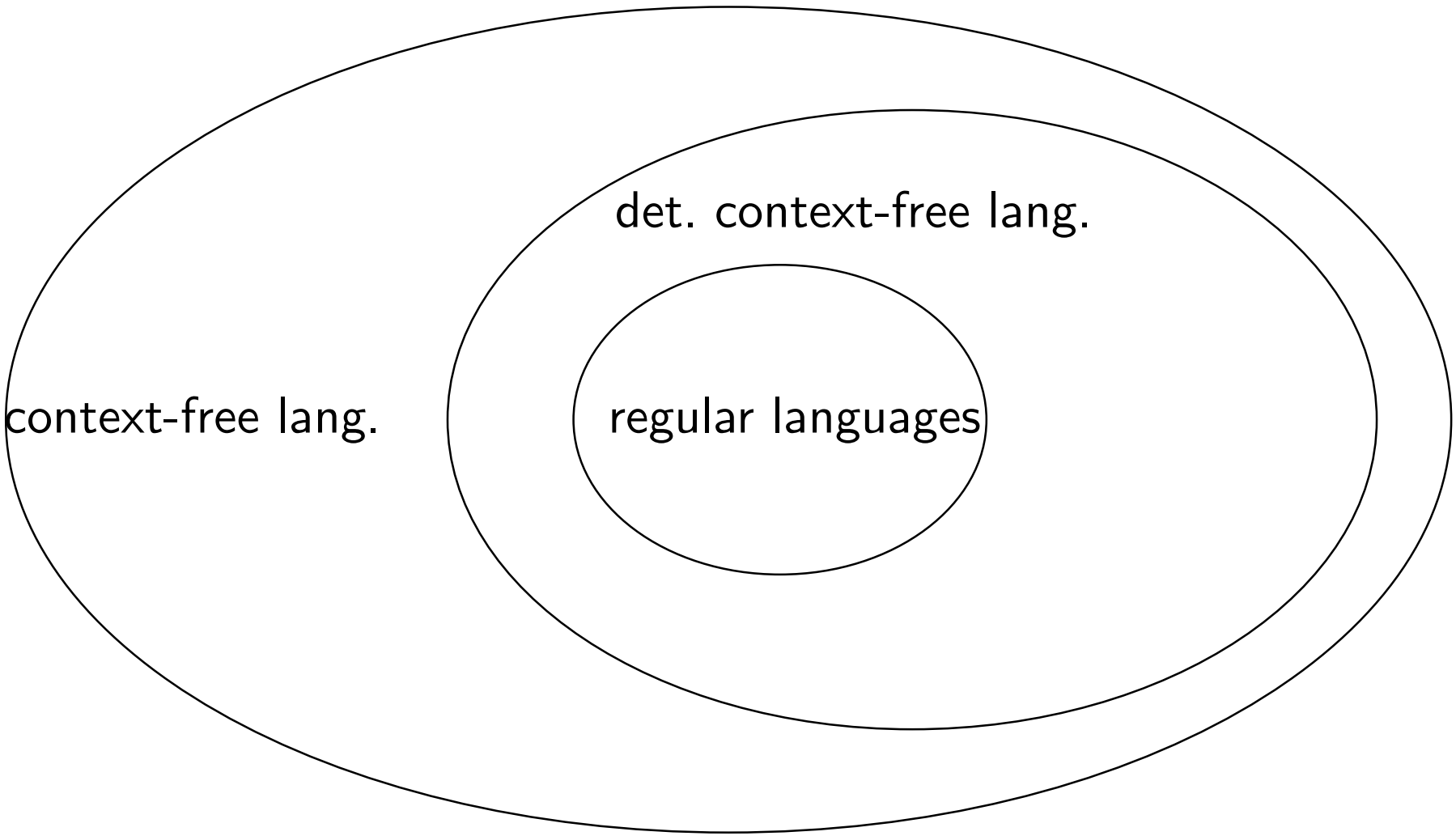
14th Lecture

Deterministic PDA

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

Deterministic PDA

Deterministic PDA



Deterministic PDA

Question

What does it formally mean that a PDA is deterministic?

Deterministic PDA

Definition

A PDA is defined to be *deterministic* (DPDA) if for all $z \in Z$, $a \in \Sigma$ and $A \in \Gamma$ it holds that

$$|\delta(z, a, A)| + \underbrace{|\delta(z, \varepsilon, A)|}_{\varepsilon \text{ transitions}} \leq 1$$

By convention, a DPDA always accepts with final states.

Deterministic PDA

Definition

A PDA is defined to be *deterministic* (DPDA) if for all $z \in Z$, $a \in \Sigma$ and $A \in \Gamma$ it holds that

$$|\delta(z, a, A)| + \underbrace{|\delta(z, \varepsilon, A)|}_{\varepsilon \text{ transitions}} \leq 1$$

By convention, a DPDA always accepts with final states.

For DPDA the languages that can be accepted with final states do not(!) equal to the languages that can be accepted by an empty stack (without proof).

Pushdown Automaton

Theorem

The set of languages that can be generated by PDAs equals to the context-free languages

Pushdown Automaton

Theorem

The set of languages that can be generated by PDAs equals to the context-free languages

Within this lecture we will only show one direction, i.e. how any context-free grammar can be converted into a PDA accepting the same language. The other direction is proven in the book of *Schöning*.

Convert CFG to PDA

Let $G = (V, \Sigma, P, S)$ be a context-free grammar for a language L. Then we create a PDA M simulating the grammar G.

Convert CFG to PDA

Let $G = (V, \Sigma, P, S)$ be a context-free grammar for a language L. Then we create a PDA M simulating the grammar G.

We chose the stack alphabet to be $V \cup \Sigma$ and use S as the lowest stack symbol. When S is POPped, the PDA will accept. M is given by

$$M = (\{z\}, \Sigma, V \cup \Sigma, \delta, z, S).$$

As one can see it has a single state.

Convert CFG to PDA

The state transition function δ is defined using the productions P. For any rule $A \to \alpha \in P$ with $\alpha \in (V \cup \Sigma)^*$ we set

$$\delta(z, \varepsilon, A) \ni (z, \alpha).$$

We further set

$$\delta(z, a, a) \ni (z, \varepsilon).$$

Convert CFG to PDA

The state transition function δ is defined using the productions P. For any rule $A \to \alpha \in P$ with $\alpha \in (V \cup \Sigma)^*$ we set

$$\delta(z, \varepsilon, A) \ni (z, \alpha).$$

We further set

$$\delta(z, a, a) \ni (z, \varepsilon).$$

The automaton will apply a P rule, if the highest stack symbol is a variable. If the highest stack symbol is a terminal symbol, it will read the same character in the input string. Here, the characters are compared so that the input string has to match the string generated on the stack. The read character on the stack will be POPed so that the stack is reduced in the end.

Convert CFG to PDA

Now for a string $x \in \Sigma^*$ we have

Convert CFG to PDA

Now for a string $x \in \Sigma^*$ we have

$x \in L(G)$

Convert CFG to PDA

Now for a string $x \in \Sigma^*$ we have

$x \in L(G)$

$\Leftrightarrow \exists$ a derivation of G of the form $S \Rightarrow \cdots \Rightarrow x$

Convert CFG to PDA

Now for a string $x \in \Sigma^*$ we have

$x \in L(G)$

$\Leftrightarrow \exists$ a derivation of G of the form $S \Rightarrow \cdots \Rightarrow x$

$\Leftrightarrow \exists$ a sequence of configurations of M of the form $(z, x, S) \vdash \cdots \vdash (z, \varepsilon, \varepsilon)$

Convert CFG to PDA

Now for a string $x \in \Sigma^*$ we have

$x \in L(G)$

$\Leftrightarrow \exists$ a derivation of G of the form $S \Rightarrow \cdots \Rightarrow x$

$\Leftrightarrow \exists$ a sequence of configurations of M of the form $(z, x, S) \vdash \cdots \vdash (z, \varepsilon, \varepsilon)$

$\Leftrightarrow x \in L(M)$

Example: CFG to PDA Conversion

Let the context free grammar

$$G = (V, \Sigma, P, S)$$
$$P = \{S \to ab \,|\, aSb\}$$

with $\Sigma = \{a, b\}$ be given.

Example: CFG to PDA Conversion

Let the context free grammar

$$G = (V, \Sigma, P, S)$$
$$P = \{S \to ab \,|\, aSb\}$$

with $\Sigma = \{a, b\}$ be given.

Then the constructed PDA looks like

$$M = (\{z\}, \Sigma, \{a, b, S\}, \delta, z, S)$$
$$1. : \delta(z, \varepsilon, S) \ni (z, ab)$$
$$2. : \delta(z, \varepsilon, S) \ni (z, aSb)$$
$$3. : \delta(z, a, a) \ni (z, \varepsilon)$$
$$4. : \delta(z, b, b) \ni (z, \varepsilon)$$

Example: CFG to PDA Conversion

Input string *aaabbb*.

Example: CFG to PDA Conversion

Input string *aaabbb*.

Configuration sequence:

$$
\begin{aligned}
(z, aaabbb, S) &\overset{2.}{\vdash} (z, aaabbb, aSb) \overset{3.}{\vdash} (z, aabbb, Sb) \\
&\overset{2.}{\vdash} (z, aabbb, aSbb) \overset{3.}{\vdash} (z, abbb, Sbb) \\
&\overset{1.}{\vdash} (z, abbb, abbb) \overset{3.}{\vdash} (z, bbb, bbb) \\
&\overset{3\times 4.}{\vdash} (z, \varepsilon, \varepsilon)
\end{aligned}
$$

PDA with Single State

Theorem

Any PDA can be converted to an equivalent PDA that has only a single state

PDA with Single State

Theorem

Any PDA can be converted to an equivalent PDA that has only a single state

Proof

Convert the PDA M into a CFG G (not shown). Then convert the grammar G into a PDA M' using the conversion discussed before. Since the conversion generated a PDA having only a single state, M' fulfills the condition stated in the theorem.

Complexity of Decision Problem

Theorem

The decision problem for deterministic context-free languages can be solved in linear time

Complexity of Decision Problem

Theorem

The decision problem for deterministic context-free languages can be solved in linear time

Proof

We consider a DPDA for the language, which reads in each step either a character or the ε sign. Since the number of states and the δ function are finite, the automaton can only require a constant number of operations to place the read pointer to the next character. If we thus consider a constant number per character, we end up at linear parsing time.

Complexity of Decision Problem

Remark

Since the deterministic context-free languages can be parsed in linear time, they are the most important class for the design of programming languages.