

# Theory of Computer Science

## C5. Context-free Languages: Normal Form and PDA

Gabriele Röger

University of Basel

April 1, 2020

# Theory of Computer Science

## April 1, 2020 — C5. Context-free Languages: Normal Form and PDA

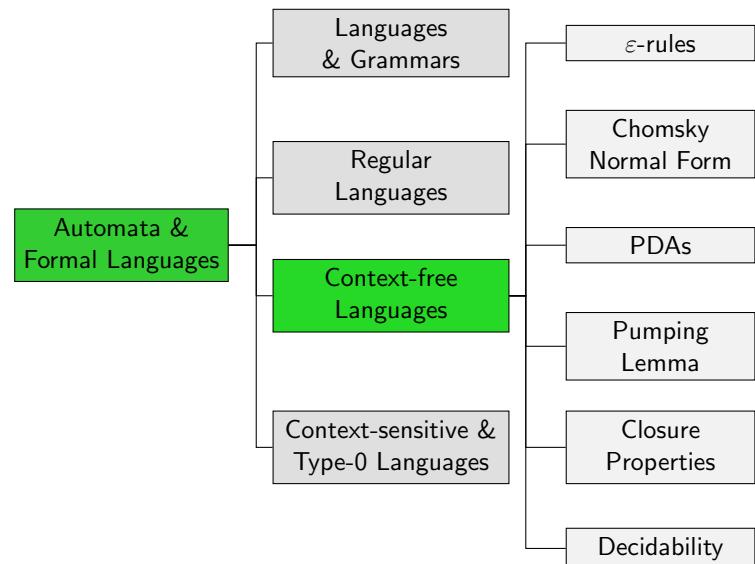
### C5.1 Context-free Grammars and $\varepsilon$ -Rules

### C5.2 Chomsky Normal Form

### C5.3 Push-Down Automata

### C5.4 Summary

## Overview



## C5.1 Context-free Grammars and $\varepsilon$ -Rules

## Repetition: Context-free Grammars

### Definition (Context-free Grammar)

A **context-free grammar** is a 4-tuple  $\langle \Sigma, V, P, S \rangle$  with

- ①  $\Sigma$  finite alphabet of terminal symbols,
- ②  $V$  finite set of variables (with  $V \cap \Sigma = \emptyset$ ),
- ③  $P \subseteq (V \times (V \cup \Sigma)^+)^* \cup \{\langle S, \varepsilon \rangle\}$  finite set of rules,
- ④ If  $S \rightarrow \varepsilon \in P$ , then all other rules in  $V \times ((V \setminus \{S\}) \cup \Sigma)^*$ .
- ⑤  $S \in V$  start variable.

Rule  $X \rightarrow \varepsilon$  is only allowed if  $X = S$  and  $S$  never occurs on a right-hand side.

With regular grammars, this restriction could be lifted.  
How about context-free grammars?

## Reminder: Start Variable in Right-Hand Side of Rules

For every type-0 language  $L$  there is a grammar where the start variable does not occur on the right-hand side of any rule.

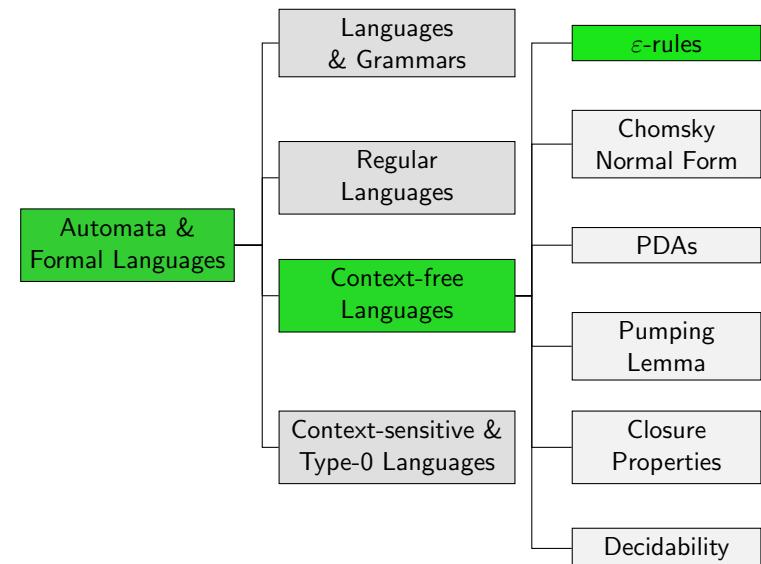
### Theorem

For every grammar  $G = \langle \Sigma, V, P, S \rangle$  there is a grammar  $G' = \langle \Sigma, V', P', S \rangle$  with rules  $P' \subseteq (V' \cup \Sigma)^* \times (V' \setminus \{S\} \cup \Sigma)^*$  such that  $\mathcal{L}(G) = \mathcal{L}(G')$ .

In the proof we constructed a suitable grammar, where the rules in  $P'$  were not fundamentally different from the rules in  $P$ :

- ▶ for rules from  $V \times (V \cup \Sigma)^+$ , we only introduced additional rules from  $V' \times (V' \cup \Sigma)^+$ , and
- ▶ for rules from  $V \times \varepsilon$ , we only introduced rules from  $V' \times \varepsilon$ , where  $V' = V \cup \{S'\}$  for some new variable  $S' \notin V$ .

## Overview



## $\varepsilon$ -Rules

### Theorem

For every grammar  $G$  with rules  $P \subseteq V \times (V \cup \Sigma)^*$  there is a context-free grammar  $G'$  with  $\mathcal{L}(G) = \mathcal{L}(G')$ .

### Proof.

Let  $G = \langle \Sigma, V, P, S \rangle$  be a grammar with  $P \subseteq V \times (V \cup \Sigma)^*$ .

Let  $G' = \langle \Sigma, V', P', S \rangle$  be a grammar with  $\mathcal{L}(G) = \mathcal{L}(G')$  with  $P' \subseteq V' \times ((V' \setminus S) \cup \Sigma)^*$ .

Let  $V_\varepsilon = \{A \in V' \mid A \xrightarrow{G'} \varepsilon\}$ . We can find this set  $V_\varepsilon$  by first collecting all variables  $A$  with rule  $A \rightarrow \varepsilon \in P'$  and then successively adding additional variables  $B$  if there is a rule  $B \rightarrow A_1 A_2 \dots A_k \in P'$  and the variables  $A_i$  are already in the set for all  $1 \leq i \leq k$ .

...

## $\varepsilon$ -Rules

### Theorem

For every grammar  $G$  with rules  $P \subseteq V \times (V \cup \Sigma)^*$   
 there is a context-free grammar  $G'$  with  $\mathcal{L}(G) = \mathcal{L}(G')$ .

### Proof (continued).

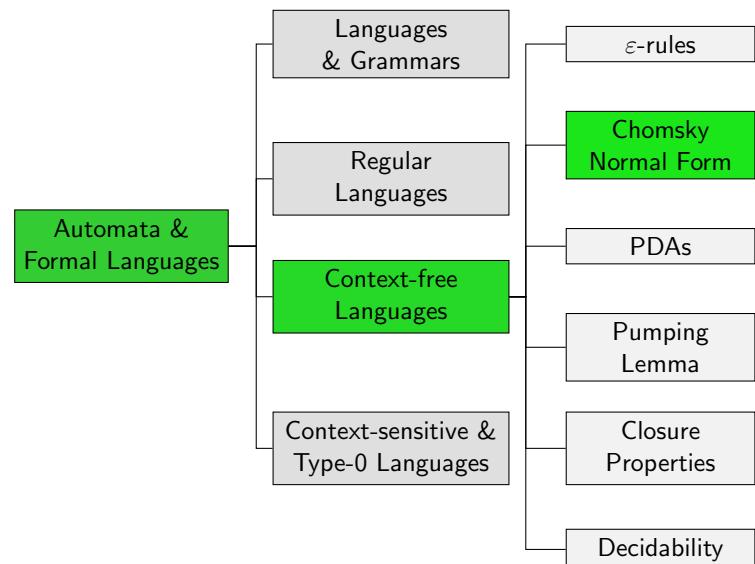
Let  $P''$  be the rule set that is constructed from  $P'$  by

- ▶ adding rules that obviate the need for  $A \rightarrow \varepsilon$  rules:  
 for every existing rule  $B \rightarrow w$  with  $B \in V'$ ,  $w \in (V' \cup \Sigma)^+$ ,  
 let  $I_\varepsilon$  be the set of positions where  $w$  contains a variable  
 $A \in V_\varepsilon$ . For every non-empty set  $I' \subseteq I_\varepsilon$ , add a new rule  
 $B \rightarrow w'$ , where  $w'$  is constructed from  $w$  by removing  
 the variables at all positions in  $I'$ .
- ▶ removing all rules of the form  $A \rightarrow \varepsilon$  ( $A \neq S$ ).

Then  $G'' = \langle \Sigma, V', P'', S \rangle$  is context-free and  $\mathcal{L}(G) = \mathcal{L}(G'')$ .  $\square$

## C5.2 Chomsky Normal Form

### Overview



### Chomsky Normal Form: Motivation

As in logical formulas (and other kinds of structured objects),  
**normal forms** for grammars are useful:

- ▶ they show which aspects are critical for defining grammars and which ones are just syntactic sugar
- ▶ they allow proofs and algorithms to be restricted to a limited set of grammars (inputs): those in normal form

Hence we now consider a **normal form** for context-free grammars.

## Chomsky Normal Form: Definition

### Definition (Chomsky Normal Form)

A context-free grammar  $G$  is in **Chomsky normal form (CNF)** if all rules have one of the following three forms:

- ▶  $A \rightarrow BC$  with variables  $A, B, C$ , or
- ▶  $A \rightarrow a$  with variable  $A$ , terminal symbol  $a$ , or
- ▶  $S \rightarrow \epsilon$  with start variable  $S$ .

German: Chomsky-Normalform

in short: rule set  $P \subseteq (V \times (VV \cup \Sigma)) \cup \{\langle S, \epsilon \rangle\}$

## Chomsky Normal Form: Theorem

### Theorem

For every context-free grammar  $G$  there is a context-free grammar  $G'$  in Chomsky normal form with  $\mathcal{L}(G) = \mathcal{L}(G')$ .

### Proof.

The following algorithm converts the rule set of  $G$  into CNF:

**Step 1: Eliminate rules of the form  $A \rightarrow B$  with variables  $A, B$ .**

If there are sets of variables  $\{B_1, \dots, B_k\}$  with rules  $B_1 \rightarrow B_2, B_2 \rightarrow B_3, \dots, B_{k-1} \rightarrow B_k, B_k \rightarrow B_1$ , then replace these variables by a new variable  $B$ .

Define a strict total order  $<$  on the variables such that  $A \rightarrow B \in P$  implies that  $A < B$ . Iterate from the largest to the smallest variable  $A$  and eliminate all rules of the form  $A \rightarrow B$  while adding rules  $A \rightarrow w$  for every rule  $B \rightarrow w$  with  $w \in (V \cup \Sigma)^+$ . . . .

## Chomsky Normal Form: Theorem

### Theorem

For every context-free grammar  $G$  there is a context-free grammar  $G'$  in Chomsky normal form with  $\mathcal{L}(G) = \mathcal{L}(G')$ .

### Proof (continued).

**Step 2: Eliminate rules with terminal symbols on the right-hand side that do not have the form  $A \rightarrow a$ .**

For every terminal symbol  $a \in \Sigma$  add a new variable  $A_a$  and the rule  $A_a \rightarrow a$ .

Replace all terminal symbols in all rules that do not have the form  $A \rightarrow a$  with the corresponding newly added variables. . . .

## Chomsky Normal Form: Theorem

### Theorem

For every context-free grammar  $G$  there is a context-free grammar  $G'$  in Chomsky normal form with  $\mathcal{L}(G) = \mathcal{L}(G')$ .

### Proof (continued).

**Step 3: Eliminate rules of the form  $A \rightarrow B_1 B_2 \dots B_k$  with  $k > 2$**

For every rule of the form  $A \rightarrow B_1 B_2 \dots B_k$  with  $k > 2$ , add new variables  $C_2, \dots, C_{k-1}$  and replace the rule with

$$A \rightarrow B_1 C_2$$

$$C_2 \rightarrow B_2 C_3$$

⋮

$$C_{k-1} \rightarrow B_{k-1} B_k$$



## Chomsky Normal Form: Length of Derivations

### Observation

Let  $G$  be a grammar in Chomsky normal form, and let  $w \in \mathcal{L}(G)$  be a non-empty word generated by  $G$ . Then all derivations of  $w$  have exactly  $2|w| - 1$  derivation steps.

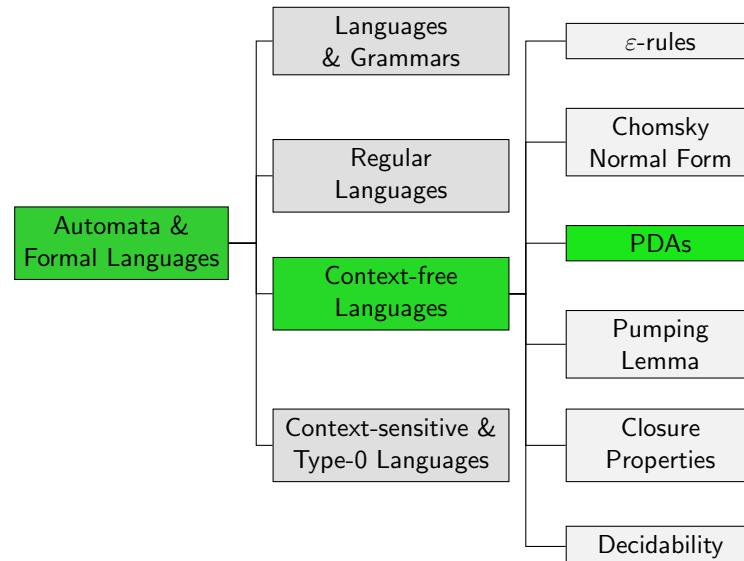
### Proof.

~ Exercises

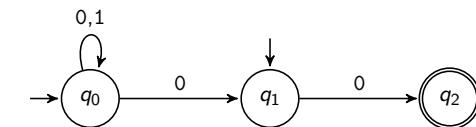


## C5.3 Push-Down Automata

## Overview



## Limitations of Finite Automata



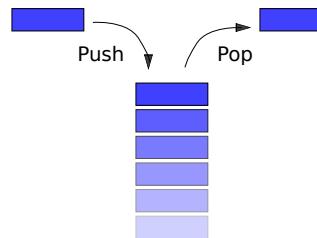
- ▶ Language  $L$  is regular.  
 $\iff$  There is a finite automaton that accepts  $L$ .
- ▶ What information can a finite automaton “store” about the already read part of the word?
- ▶ Infinite memory would be required for  $L = \{x_1x_2 \dots x_nx_n \dots x_2x_1 \mid n > 0, x_i \in \{a, b\}\}$ .
- ▶ therefore: extension of the automata model with memory

## Stack

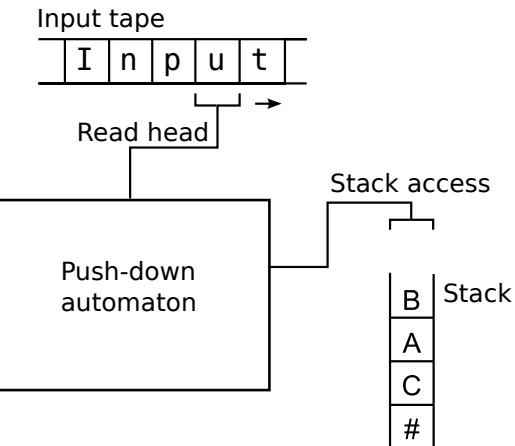
A **stack** is a data structure following the **last-in-first-out (LIFO)** principle supporting the following operations:

- ▶ **push**: puts an object on top of the stack
- ▶ **pop**: removes the object at the top of the stack
- ▶ **peek**: returns the top object without removing it

German: Keller, Stapel



## Push-down Automata: Visually



German: Kellerautomat, Eingabeband, Lesekopf, Kellerzugriff

## Push-down Automata: Definition

### Definition (Push-down Automaton)

A **push-down automaton (PDA)** is a 6-tuple  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \# \rangle$  with

- ▶  $Q$  finite set of states
- ▶  $\Sigma$  the input alphabet
- ▶  $\Gamma$  the stack alphabet
- ▶  $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \mathcal{P}_f(Q \times \Gamma^*)$  the transition function (where  $\mathcal{P}_f$  is the set of all **finite** subsets)
- ▶  $q_0 \in Q$  the start state
- ▶  $\# \in \Gamma$  the bottommost stack symbol

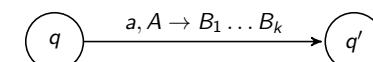
German: Kellerautomat, Eingabealphabet, Kelleralphabet, Überführungsfunktion

## Push-down Automata: Transition Function

Let  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \# \rangle$  be a push-down automaton.

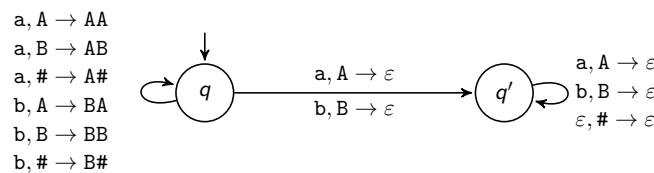
### What is the Intuitive Meaning of the Transition Function $\delta$ ?

- ▶  $\langle q', B_1 \dots B_k \rangle \in \delta(q, a, A)$ : If  $M$  is in state  $q$ , reads symbol  $a$  and has  $A$  as the topmost stack symbol, then  $M$  **can** transition to  $q'$  in the next step while replacing  $A$  with  $B_1 \dots B_k$  (afterwards  $B_1$  is the topmost stack symbol)



- ▶ special case  $a = \epsilon$  is allowed (spontaneous transition)

## Push-down Automata: Example



$M = \langle \{q, q'\}, \{a, b\}, \{A, B, \#\}, \delta, q, \# \rangle$  with

$$\begin{array}{lll}
 \delta(q, a, A) = \{\langle q, AA \rangle, \langle q', \epsilon \rangle\} & \delta(q, b, A) = \{\langle q, BA \rangle\} & \delta(q, \epsilon, A) = \emptyset \\
 \delta(q, a, B) = \{\langle q, AB \rangle\} & \delta(q, b, B) = \{\langle q, BB \rangle, \langle q', \epsilon \rangle\} & \delta(q, \epsilon, B) = \emptyset \\
 \delta(q, a, \#) = \{\langle q, A\# \rangle\} & \delta(q, b, \#) = \{\langle q, B\# \rangle\} & \delta(q, \epsilon, \#) = \emptyset \\
 \delta(q', a, A) = \{\langle q', \epsilon \rangle\} & \delta(q', b, A) = \emptyset & \delta(q', \epsilon, A) = \emptyset \\
 \delta(q', a, B) = \emptyset & \delta(q', b, B) = \{\langle q', \epsilon \rangle\} & \delta(q', \epsilon, B) = \emptyset \\
 \delta(q', a, \#) = \emptyset & \delta(q', b, \#) = \emptyset & \delta(q', \epsilon, \#) = \{\langle q', \epsilon \rangle\}
 \end{array}$$

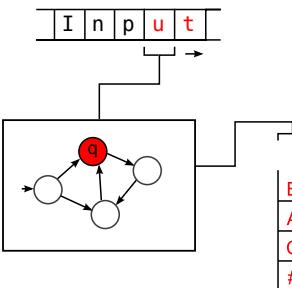
## Push-down Automata: Configuration

### Definition (Configuration of a Push-down Automaton)

A **configuration** of a push-down automaton  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \# \rangle$  is given by a triple  $c \in Q \times \Sigma^* \times \Gamma^*$ .

German: Konfiguration

### Example



Configuration  
 $\langle q, ut, BAC\# \rangle$ .

## Push-down Automata: Steps

### Definition (Transition/Step of a Push-down Automaton)

We write  $c \vdash_M c'$  if a push-down automaton  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \# \rangle$  can transition from configuration  $c$  to configuration  $c'$  in one step. Exactly the following transitions are possible:

$$\langle q, a_1 \dots a_n, A_1 \dots A_m \rangle \vdash_M \begin{cases} \langle q', a_2 \dots a_n, B_1 \dots B_k A_2 \dots A_m \rangle \\ \text{if } \langle q', B_1 \dots B_k \rangle \in \delta(q, a_1, A_1) \\ \langle q', a_1 a_2 \dots a_n, B_1 \dots B_k A_2 \dots A_m \rangle \\ \text{if } \langle q', B_1 \dots B_k \rangle \in \delta(q, \epsilon, A_1) \end{cases}$$

German: Übergang

If  $M$  is clear from context, we only write  $c \vdash c'$ .

## Push-down Automata: Reachability of Configurations

### Definition (Reachable Configuration)

Configuration  $c'$  is **reachable** from configuration  $c$  in PDA  $M$  ( $c \vdash_M^* c'$ ) if there are configurations  $c_0, \dots, c_n$  ( $n \geq 0$ ) where

- $c_0 = c$ ,
- $c_i \vdash_M c_{i+1}$  for all  $i \in \{0, \dots, n-1\}$ , and
- $c_n = c'$ .

German:  $c'$  ist in  $M$  von  $c$  erreichbar

## Push-down Automata: Recognized Words

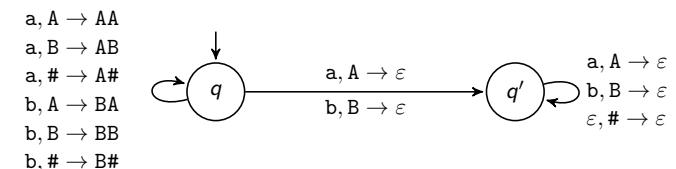
### Definition (Recognized Word of a Push-down Automaton)

PDA  $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \# \rangle$  **recognizes the word**  $w = a_1 \dots a_n$  iff the configuration  $\langle q, \varepsilon, \varepsilon \rangle$  (word processed and stack empty) for some  $q \in Q$  is reachable from the **start configuration**  $\langle q_0, w, \# \rangle$ .

$M$  recognizes  $w$  iff  $\langle q_0, w, \# \rangle \vdash_M^* \langle q, \varepsilon, \varepsilon \rangle$  for some  $q \in Q$ .

German:  $M$  erkennt  $w$ , Startkonfiguration

## Push-down Automata: Recognized Word Example



example: this PDA recognizes  $bbabbabb \rightsquigarrow$  blackboard

## Push-down Automata: Accepted Language

### Definition (Accepted Language of a Push-down Automaton)

Let  $M$  be a push-down automaton with input alphabet  $\Sigma$ .

The **language accepted by  $M$**  is defined as

$$\mathcal{L}(M) = \{w \in \Sigma^* \mid M \text{ recognizes } w\}.$$

example: blackboard

## PDAs Accept Exactly the Context-free Languages

### Theorem

A language  $L$  is context-free if and only if  $L$  is accepted by a push-down automaton.

## C5.4 Summary

## Summary

- ▶ Every context-free language has a grammar in **Chomsky normal form**. All rules have form
  - ▶  $A \rightarrow BC$  with variables  $A, B, C$ , or
  - ▶  $A \rightarrow a$  with variable  $A$ , terminal symbol  $a$ , or
  - ▶  $S \rightarrow \epsilon$  with start variable  $S$ .
- ▶ **Push-down automata** (PDAs) extend NFAs with memory.
- ▶ PDAs **accept** not with end states but with an **empty stack**.
- ▶ The **languages accepted by PDAs** are exactly the **context-free languages**.