# Automata and Grammars (BIE-AAG) 1. Basic notions

### Jan Holub

Department of Theoretical Computer Science Faculty of Information Technology Czech Technical University in Prague



© Jan Holub, 2020

# **Motivation**

#### **Motivation**

- Theory of languages and automata is one of the building blocks of theoretical computer science.
- The theory offers efficient solutions to a number of basic problems.
- The right use of the theory saves time and money.
- Applications: compiler construction, model checking, verification of communication protocols, pattern matching, data compression, . . .

# **Course goals**

# **Course goals**

- To get to know the languages and automata theory.
- To be able to use the theory efficiently.
- To recognize classes of languages.

# **Course evaluation**

### **Seminars**

- homeworks: one for 5 points, the others for 0 points each (Student's responsibility.)
- 2 tests (7th week: Marast online test for 10 pts; 11th week: contact test for 25 pts)
- up to 5 points for activity at seminars
- 40 points in total
- assessment ("zápočet"): 20 points at minimum

# **Course evaluation**

#### Exam

- "multiple choice": 10 points at max.
- main test: 50 points at max.
- requirements: multiple choice min. 7 points, main test min. 25 points (successfully completed multiple choice test is considered in the next exam)

# Course schedule

- 1. Basic notions, Chomsky hierarchy, Closure properties of CFL
- 2. Deterministic and nondeterministic finite automata (DFA and NFA), NFA with epsilon transitions
- 3. Operations on automata (removal of epsilon transitions, determinization, minimization, intersection, union)
- 4. Reg. expressions, RE, FA and RG conversions, Kleene theorem
- 5. Operations on regular grammars, conversions to FA
- 6. Properties of regular languages (pumping lemma, Nerode th.)
- 7. Context-free languages, pushdown automaton
- 8. Parsing of Context-free languages (nondet. vs. determinism)
- 9. Transducers, Mealey, Moore, conversions
- 10. Context-sensitive and Recursively enumerable languages, Turing machine
- 11. Program and circuit implementation of DFA and NFA
- 12. FA as a lexical analyzer, lex/flex generator

# Literature

- Aho A. V, Motwani R., Ullman, J. D.: Introduction to Automata Theory, Languages, and Computation. (2nd Edition). Addison Wesley, 2001. ISBN 0-201-44124-1.
- Kozen, D. C.: Automata and Computability. Springer, 1997. ISBN 0387949070.
- Melichar B., Holub J., Mužátko P.: Languages and Translations. Praha: Publishing House of CTU, 1997. ISBN 80-01-01692-7.
- Šestáková E.: Automata and Grammars. A Collection of Excercises and Solutions. Praha, Pražská technika – Nakladatelství ČVUT, 2018.

# Warning

The slides are only auxiliary materials for lectures and cannot be considered as the only study source for tests and exam.

# **Basic notions**

Alphabet – finite set of symbols (notation:  $\Sigma$  or T)

■ binary {0,1}, ternary {Yes, No, Maybe},
DNA {A, C, G, T}, keywords {while, do, begin, end, to, for, false, true, ...}

String over an alphabet – a finite sequence of symbols of the alphabet. E.g. "0110101", "ACCCGT", "while true do"

Empty sequence = *empty string* =  $\varepsilon$ .

 $\Sigma^*$  – set of all strings over  $\Sigma$ 

 $\Sigma^+$  – set of all nonempty strings over  $\Sigma$ 

$$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$$

# **Basic notions**

Operation concatenation (denoted as '.'):

- $\forall x,y \in \Sigma^*$ , string x.y (shortened to xy) is created by concatenating strings x and y.
- $\blacksquare$  is associative, i.e.,  $\forall x,y,z\in\Sigma^*:(xy)z=x(yz)$ ,
- $\blacksquare$  is not commutative, i.e.,  $\exists x,y \in \Sigma^* : xy \neq yx$ ,
- lacksquare acts as the identity element of concatenation operation: xarepsilon=arepsilon x=arepsilon x=arepsilon
- $a^0 = \varepsilon, \ a^1 = a, \ a^2 = aa, \ a^3 = aaa, \dots$

Operation reversal of string (denoted as  $x^R$ ):

- $\mathbf{I}$   $x = a_1 a_2 a_3 \dots a_n, \ x^R = a_n a_{n-1} \dots a_1$
- $y = abcd, y^R = dcba$

# **Basic notions**

Length of a string x:

- lacksquare denoted by |x|
- $|x| \ge 0,$

# Formal language

Formal language L over  $\Sigma$ :  $L \subseteq \Sigma^*$  (a set of strings).

### operations:

- set operations: *union*, *intersection*, *difference*
- **omplement** of language  $L_1$ :  $\overline{L_1} = \Sigma^* \setminus L_1$   $(\overline{L_1} \cup L_1 = \Sigma^*, L_1 \cap \overline{L_1} = \emptyset)$ .
- concatenation (product) of languages:  $L = L_1.L_2 = \{xy : x \in L_1, y \in L_2\}$  (L is defined over alphabet  $\Sigma = \Sigma_1 \cup \Sigma_2$ )
- n-th power of language L:  $L^n = L.L^{n-1}$ ,  $L^0 = \{\varepsilon\}$ . Kleene star  $L^*$  of language L:  $L^* = \bigcup_{n=0}^{\infty} L^n$ .  $L^* = L^+ \cup \{\varepsilon\}$ ,  $L^+ = L.L^* = L^*.L = \bigcup_{n=1}^{\infty} L^n$  (Kleene plus)

#### **Definition**

*Grammar* is a quadruple  $G = (N, \Sigma, P, S)$ , where

- lacksquare N is a finite set of nonterminal symbols,
- lacksquare  $\Sigma$  is a finite set of terminal symbols  $(\Sigma \cap N = \emptyset$ , denoted also by T),
- P is a set of *production rules*. It is a finite subset of  $(N \cup \Sigma)^*.N.(N \cup \Sigma)^* \times (N \cup \Sigma)^*$ , (element  $(\alpha, \beta)$  of P is written as  $\alpha \to \beta$  and called a *rule*),
- lacksquare  $S \in N$  is the *start symbol* of the grammar.

# **Example**

Grammar  $G_1 = (\{A, S\}, \{0, 1\}, P, S)$ , where P:

- $\blacksquare$   $S \rightarrow 0A$
- $\blacksquare$   $A \rightarrow 1A$
- $\blacksquare$   $A \rightarrow 0.$

### Note:

$$\alpha \to \beta_1, \ \alpha \to \beta_2, \dots, \alpha \to \beta_n$$
, can be shortened to:  $\alpha \to \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$ .

Other possibilities to describe grammar:

- Backus-Naur Form (BNF),
- Extended Backus-Naur Form (EBNF).

# **Example**

Grammar G generates a language of unsigned integers. BNF is used to specify this grammar.

```
G = (\{\langle \mathit{integer} \rangle, \langle \mathit{digit} \rangle\}, \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, P, \langle \mathit{integer} \rangle) Set P contains rules: \langle \mathit{integer} \rangle ::= \langle \mathit{digit} \rangle \langle \mathit{integer} \rangle \mid \langle \mathit{digit} \rangle \langle \mathit{digit} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9.
```

#### **Definition**

 $G = (N, \Sigma, P, S)$ ,  $x, y \in (N \cup \Sigma)^*$ . We say that x derives y in one step  $(x \Rightarrow y)$ , if there exists  $(\alpha \to \beta) \in P$  and  $\gamma, \delta \in (N \cup \Sigma)^*$  such that  $x = \gamma \alpha \delta, y = \gamma \beta \delta$  (i.e.,  $\gamma \alpha \delta \Rightarrow \gamma \beta \delta$ ).

### **Example**

$$G_1 = (\{A, S\}, \{0, 1\}, P, S), P = \{S \rightarrow 0A00, A \rightarrow 1A, A \rightarrow 0\}.$$
  
 $01A00 \Rightarrow 011A00 \quad (\gamma = 01, \alpha = A, \beta = 1A, \delta = 00)$ 

#### **Definition**

 $\alpha \Rightarrow^k \beta$  if there exists a sequence  $\alpha_0, \alpha_1, \ldots, \alpha_k$  for  $k \geq 0$ , of k+1 strings such that  $\alpha = \alpha_0, \alpha_{i-1} \Rightarrow \alpha_i$  for  $1 \leq i \leq k$ , and  $\alpha_k = \beta$ . This sequence is called *derivation* of string  $\beta$  from string  $\alpha$  that has length k in grammar G.

### **Example**

$$G_1: 01A00 \Rightarrow^4 011111A00$$

#### **Definition**

Transitive closure of relation  $\Rightarrow$ :  $\alpha \Rightarrow^+ \beta$  if  $\alpha \Rightarrow^i \beta$  for some  $i \ge 1$ . (We read ' $\Rightarrow^+$ ' as "derives in nonzero number of steps".)

#### **Definition**

Transitive and reflexive closure of relation  $\Rightarrow$ :  $\alpha \Rightarrow^* \beta$  if  $\alpha \Rightarrow^i \beta$  for some  $i \ge 0$ .

(We read  $\Rightarrow^*$  as "derives in any number of steps".)

### **Definition**

 $G=(N,\Sigma,P,S)$ . String  $\alpha$  is called a *sentential form* in grammar G, if  $S\Rightarrow^*\alpha,\alpha\in(N\cup\Sigma)^*$ .

# **Example**

 $G_1: S \Rightarrow^* 01A00$ 

 $G_1: S \Rightarrow^* 011000$ 

### **Definition**

A sentential form in  $G=(N,\Sigma,P,S)$  that contains no nonterminal symbols is called a *sentence generated by grammar* G.

### **E**xample

 $G_1: S \Rightarrow^* 011000$ 

#### **Definition**

 $L(G)=\{w:w\in\Sigma^*,\exists S\Rightarrow^*w\}$  is the language generated by grammar  $G=(N,\Sigma,P,S).$ 

(Language generated by grammar G is the set of all sentences generated by grammar G.)

### Remark

Note that  $\varepsilon$  is a sentence generated by grammar G, if  $\varepsilon \in L(G)$ .

### **Example**

$$G_1 = (\{A, S\}, \{0, 1\}, \{S \to 0A, A \to 1A, A \to 0\}, S)$$
 generates language  $L(G_1) = \{01^n0 : n \ge 0\}.$ 

The following derivations exist in grammar  $G_1$ :

$$S \Rightarrow 0A \Rightarrow 00$$
  
 $S \Rightarrow 0A \Rightarrow 01A \Rightarrow 010$   
 $S \Rightarrow 0A \Rightarrow 01A \Rightarrow 011A \Rightarrow 0110$ 

### **Definition**

Grammars  $G_1$  and  $G_2$  are equivalent if they generate the same language. That means  $L(G_1) = L(G_2)$ .

# Classification of grammars

Noam Chomsky (\*7.12.1928 Philadelphia)

work in the field of grammars of both formal and natural languages

#### **Definition**

 $G = (N, \Sigma, P, S)$ . We say that G is:

- 0. Unrestricted (type 0), if it satisfies the general grammar definition.
- 1. Context-sensitive (type 1), if every rule from P is of the form  $\gamma A \ \delta \to \gamma \alpha \delta$ , where  $\gamma, \delta \in (N \cup \Sigma)^*, \alpha \in (N \cup \Sigma)^+, A \in N$ , or the form  $S \to \varepsilon$  in case that S is not present in the right-hand side of any rule.
- 2. Context-free (type 2), if every rule is of the form  $A \to \alpha$ , where  $A \in N, \alpha \in (N \cup \Sigma)^*$ .
- 3. Regular (type 3), if every rule is of the form  $A \to aB$  or  $A \to a$ , where  $A, B \in N, a \in \Sigma$ , or the form  $S \to \varepsilon$  in case that S is not present in the right-hand side of any rule.

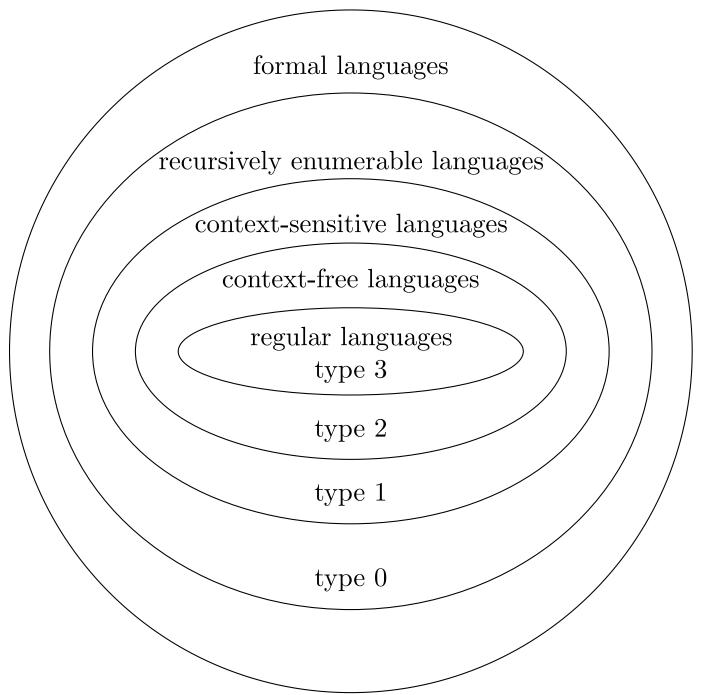
# Classification of languages

#### **Definition**

We say that language

- 0. is recursively enumerable (type 0), if  $\exists$  unrestricted grammar which generates it.
  - recognized by Turing machine.
- 1. is context-sensitive (type 1), if  $\exists$  context-sensitive grammar which generates it.
  - recognized by linear bounded Turing machine (also called linear bounded automaton).
- 2. is context-free (type 2), if  $\exists$  context-free grammar which generates it.
  - recognized by nondeterministic pushdown automaton.
- 3. is regular (type 3), if  $\exists$  regular grammar which generates it.
  - recognized by finite automaton.

# Classification of languages



# Classification of languages

### **Example**

 $G_1 = (\{S\}, \{0, 1\}, \{S \to 0S, S \to 1S, S \to 1, S \to 0\}, S)$  is a regular grammar and generates language  $L(G_1) = \{0, 1\}^+$ .

 $G_2 = (\{S\}, \{0, 1\}, \{S \to 0S1, S \to 01\}, S)$  is context-free and generates language  $L(G_2) = \{0^n 1^n : n \ge 1\}$ .

 $G_3 = (\{S, A\}, \{0, 1\}, \{S \to 0A1, S \to 01, 0A \to 00A1, A \to 01\}, S)$  is context-sensitive and generates  $L(G_3) = \{0^n 1^n : n \ge 1\}$ .

 $G_4 = (\{S\}, \{0, 1\}, \{S \to 0S1, S \to 01, 0S1 \to S\}, S)$  is unrestricted and generates language  $L(G_4) = \{0^n 1^n : n \ge 1\}$ .

Grammars  $G_2, G_3$  and  $G_4$  are equivalent, because  $L(G_2) = L(G_3) = L(G_4) = \{0^n 1^n : n \ge 1\}.$ 

 $L(G_1)$  is regular language,  $L(G_2)$  is not, but  $L(G_2) \subseteq L(G_1)$ .

# **Closure Properties of CFL**

### Closure property:

If certain languages are context-free, and a language L is formed from them by certain operations, then L is also context-free.

#### **Theorem**

The class of context-free languages is closed under operations union, product, and iteration.

# Grammars and ops. over languages

**Algorithm** Grammar for a *union* of languages.

**Input:** Context-free grammars  $G_1=(N_1,\Sigma,P_1,S_1)$  and  $G_2=(N_2,\Sigma,P_2,S_2)$  generating languages  $L_1$  and  $L_2$ ,  $N_1\cap N_2=\emptyset$ .

**Output:** Context-free grammar G,  $L(G) = L_1 \cup L_2$ .

1: 
$$G \leftarrow (N_1 \cup N_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \rightarrow S_1 \mid S_2\}, S)$$
, where  $S \notin N_1 \cup N_2$ .

W.L.O.G., the sets of terminal symbols are expected to be equal.

# Grammars and ops. over languages

Algorithm Grammar for a product of languages.

**Input:** Context-free grammars  $G_1 = (N_1, \Sigma, P_1, S_1)$  and

 $G_2=(N_2,\Sigma,P_2,S_2)$  generating languages  $L_1$  and  $L_2$ ,  $N_1\cap N_2=\emptyset$ .

**Output:** Context-free grammar G such that  $L(G) = L_1.L_2$ .

1:  $G \leftarrow (N_1 \cup N_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \rightarrow S_1S_2\}, S)$ , where  $S \notin N_1 \cup N_2$ .

# Grammars and ops. over languages

**Algorithm** Grammar for a *Kleene star* of a language.

**Input:** Context-free grammar  $G = (N, \Sigma, P, S)$  generating language L.

**Output:** Context-free grammar G' such that  $L(G') = L^*$ .

1:  $G' \leftarrow (N \cup \{S'\}, \Sigma, P \cup \{S' \rightarrow SS', S' \rightarrow \varepsilon\}, S')$ , where  $S' \notin N$ .

# Finite languages

#### **Definition**

Language  $L \subset \Sigma^*$  is called *finite* if  $\exists n \in \mathbb{N}$ : |L| < n.

#### **Theorem**

The smallest class of languages that contains all finite languages and languages created from finite languages by a finite number of operations

- a) union,
- b) product,
- c) Kleene star,
- d) complement,

is the set (class) of regular languages.

# Parse tree for CF languages

A parse tree is a graphical representation of a syntactical structure of a sentential form.

# Parse tree for CF languages

### **Definition**

Given grammar  $G = (N, \Sigma, P, S)$ . Parse tree is an tree having the following properties:

- 1. The nodes of the parse tree are labeled with terminal symbol, nonterminal symbols, and symbol  $\varepsilon$  (provided it is the only child node of his parent node).
- 2. The root of the tree is labeled with the start symbol S.
- 3. If a node has at least one descendant, it is labeled by a nonterminal symbol.
- 4. If  $n_1, n_2, \ldots, n_k$  are the direct descendants of node n that is labeled with symbol A and these symbols are (from left to right) labeled with symbols  $A_1, A_2, \ldots, A_k$ , then  $A \to A_1 A_2 \ldots A_k$  is a rule in P.
- 5. The leaves of the parse tree form, from left to right, a sentential form or a sentence in grammar G, which is the *result* of the parse tree.

# Parse tree for CF languages

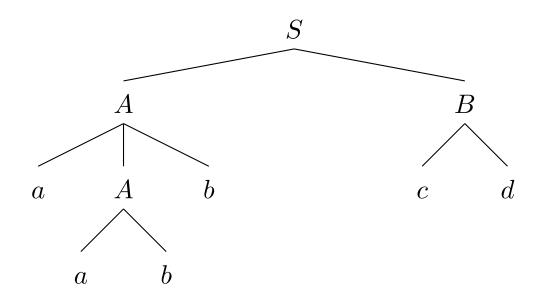
### **Example**

 $G = (\{S, A, B\}, \{a, b, c, d\}, P, S)$ , where P:

- (1)  $S \to AB$  (2)  $A \to aAb$  (3)  $A \to ab$

- (4)  $B \rightarrow cBd$  (5)  $B \rightarrow cd$

 $S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbcd.$ 



$$S \Rightarrow AB \Rightarrow Acd \Rightarrow aAbcd \Rightarrow aabbcd$$
  
 $S \Rightarrow AB \Rightarrow aAbB \Rightarrow aAbcd \Rightarrow aabbcd$