

# Formal Languages and Automata Theory

Prof. Dr.-Ing. Sebastian Schlesinger

Berlin School for Economics and Law

August 27, 2025

# Outline

# Formal Languages

Let  $\Sigma$  be an alphabet. A *formal language* over  $\Sigma$  is a subset of  $\Sigma^*$ , the set of all strings over  $\Sigma$ .

## Example

$\Sigma = \{ (, ), +, -, *, /, a \}$ . Then we can define the correctly formed arithmetic expressions as the formal language  $EXPR \subseteq \Sigma^*$ . For example,  $a + (a * a)$  is in  $EXPR$ , but  $a + *a$  is not.

# Grammars

A *grammar* is a set of rules for generating strings in a formal language. Formally, it is a 4-tuple  $G = (V, \Sigma, P, S)$  such that

- $V$  is a finite set of variables or non-terminal symbols.
- $\Sigma$  is a finite set of terminal symbols. It must hold  $V \cap \Sigma = \emptyset$ .
- $P$  is a finite set of production rules. Formally,  $P$  is a finite set of pairs  $(V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ .
- $S \in V$  is the start symbol.

Productions are usually written in the form  $u \rightarrow v$  where  $u \in (V \cup \Sigma)^+$  and  $v \in (V \cup \Sigma)^*$ .

# Formal Languages

Let  $u, v \in (V \cup \Sigma)^*$ . We define a relation  $u \Rightarrow_G v$  (in words:  $u$  derives  $v$  immediately in  $G$ ) if  $u$  and  $v$  have the form  $u = xyz$  and  $v = xy'z$  for

$x, z \in (V \cup \Sigma)^*$  and  $y \rightarrow y'$  is a production rule in  $P$ . If  $G$  is clear, we may also just write  $u \Rightarrow v$ . The language generated by a grammar  $G$  is

$L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$  where  $\Rightarrow_G^*$  is the reflexive transitive closure of  $\Rightarrow_G$ .

## Example

$G = (\{E, T, F\}, \{ (, ), a, +, * \}, P, E)$  where  
 $P = \{E \rightarrow T, E \rightarrow E + T, T \rightarrow F, T \rightarrow T * F, F \rightarrow (E), F \rightarrow a\}$ . This  
yields the language of arithmetic expressions.

## Example

$G = (V, \Sigma, P, S)$  where  $V = \{S, B, C\}$ ,  $\Sigma = \{a, b, c\}$ ,  $P = \{S \rightarrow aSBC, S \rightarrow aBC, CB \rightarrow BC, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\}$ . In this language, for instance

$S \Rightarrow aSBC \Rightarrow aaSBCBC \Rightarrow aaaBCBCBC \Rightarrow aaaBBCCBC \Rightarrow$   
 $aaaBBCBCC \Rightarrow aaaBBBCCC \Rightarrow aaabBBCCC \Rightarrow aaabbBBCCC \Rightarrow$   
 $aaabbbCCC \Rightarrow aaaabbbccC \Rightarrow aaabbbccC \Rightarrow aaabbbccc = a^3b^3c^3$ .

In total,  $L(G) = \{a^n b^n c^n \mid n \geq 1\}$ .

# Chomsky Hierarchy

- Type 0: Recursively enumerable languages - every language without
- Type 1: Context-sensitive languages: if for all productions  $w_1 \rightarrow w_2$  it holds  $|w_1| \leq |w_2|$ .
- Type 2: Context-free languages: if for all productions  $w_1 \rightarrow w_2$  it holds  $w_1 \in V$  (meaning it is just a variable on the left side).
- Type 3: Regular languages: if for all productions  $w_1 \rightarrow w_2$  it holds  $w_1 \in V$  and  $w_2 \in \Sigma \cup \Sigma V$ , i.e., the right sides are either terminals or a terminal followed by a variable.

A language  $L \subseteq \Sigma^*$  is of type  $i$  if there exists a grammar of type  $i$  that generates  $L$ .



## $\varepsilon$ rule

For type 1,2,3 grammars, because of  $|w_1| \leq |w_2|$ , the empty word  $\varepsilon$  could not be generated. However, if  $\varepsilon \in L(G)$  is desired, we allow the rule  $S \rightarrow \varepsilon$  where  $S$  is the start symbol. This is called the  $\varepsilon$  rule.

The languages of the Chomsky hierarchy form a strict hierarchy, i.e., there are languages that are of type  $i$  but not of type  $j$  for  $i < j$ . For example, the language  $L = \{a^n b^n \mid n \geq 1\}$  is context-free but not regular.

### Example

$L = \{a^n b^n \mid n \geq 1\}$  is of type 2 but not of type 3.  $L = \{a^n b^n c^n \mid n \geq 1\}$  is of type 1 but not of type 2.  $L = H$  is of type 0 but not of type 1. Here,  $H$  is the language of the halting problem.

Languages of types 1,2,3 are decidable, i.e., there exists an algorithm that decides whether a given word is in the language or not. For type 0, this is not the case. They are semi-decidable, i.e., there exists an algorithm that decides whether a given word is in the language, but it may not terminate if the word is not in the language. Another term for type 0 languages is recursively enumerable languages.

# Word Problem

The word problem is the problem of deciding whether a given word  $w \in \Sigma^*$  is in a given language, i.e.  $w \in L(G)$  or  $w \notin L(G)$ . For regular languages, this is decidable. For context-free languages, this is also decidable. For context-sensitive languages, this is also decidable. For recursively enumerable languages, this is semi-decidable.

# Finite Automata

A finite automaton is a 5-tuple  $M = (Q, \Sigma, \delta, q_0, F)$  where

- $Q$  is a finite set of states.
- $\Sigma$  is the input alphabet. It holds  $Q \cap \Sigma = \emptyset$ .
- $\delta : Q \times \Sigma \rightarrow Q$  is the transition function.
- $q_0 \in Q$  is the start state.
- $F \subseteq Q$  is the set of accepting states.

An automaton can be drawn as a graph.

# Example

$Q = \{q_0, q_1, q_2, q_3\}$ ,  $\Sigma = \{a, b\}$ ,  $F = \{q_3\}$ ,  $q_0$  is the start state.  
 $\delta(q_0, a) = q_1$ ,  $\delta(q_1, b) = q_2$ ,  $\delta(q_2, a) = q_3$ . This automaton  
accepts the language  $\{ab^n a \mid n \geq 0\}$ .

# Accepted Languages

For a finite automaton  $M = (Q, \Sigma, \delta, q_0, F)$ ,