Formal Languages and Automata Theory

Prof. Dr.-Ing. Sebastian Schlesinger

Berlin School for Economics and Law

October 17, 2025

Formal Languages

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

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.
- Σ 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 $u \to v$ 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 $y \to y'$ 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
$P = \{E \to T, E \to E + T, T \to F, T \to T * F, F \to (E), F \to a\}$. This yields the language of arithmetic expressions.

Example

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

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

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

Chomsky Hierarchy

- Type 0: Recursively enumerable languages - every language without restrictions on the production rules.
- Type 1: Context-sensitive languages: if for all productions $w_1 \to w_2$ it holds $|w_1| \leq |w_2|$.
- Type 2: Context-free languages: if for all productions $w_1 \to w_2$ it holds $w_1 \in V$ (meaning it is just a variable on the left side).
- Type 3: Regular languages: if for all productions $w_1 \to w_2$ 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.

ε rule

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

$L = \{a^n b^n | n \geq 1\}$ is of type 2 but not of type 3. $L = \{a^n b^n c^n | n \geq 1\}$ 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.

Deterministic Finite Automata

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

- Q is a finite set of states.
- Σ is the input alphabet. It holds $Q \cap \Sigma = \emptyset$.
- $\delta : Q \times \Sigma \to Q$ 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.

DFA as Graph

- Each state $q \in Q$ is represented as a node.
- The start state q_0 has an incoming arrow from nowhere.
- Each accepting state $q \in F$ is represented as a double circle.
- For each transition $\delta(q, a) = p$, there is a directed edge from node q to node p labeled with the symbol a.

Note that δ is a total function, i.e., for each state $q \in Q$ and each symbol $a \in \Sigma$, there is exactly one transition defined.

Non-deterministic Finite Automata

A non-deterministic finite automaton (NFA) is a 5-tuple $M = (Q, \Sigma, \delta, S_0, F)$ where

- Q is a finite set of states.
- Σ is the input alphabet. It holds $Q \cap \Sigma = \emptyset$.
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ is the transition function.
- $S_0 \subseteq Q$ is the set of start states.
- $F \subseteq Q$ is the set of accepting states.

An NFA can also be drawn as a graph, similar to a DFA. Since δ a function to sets of states, there may be multiple transitions for the same state and input symbol and there may also be no transition for a given state and input symbol.

Accepted Languages

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, we define $\hat{\delta} : Q \times \Sigma^* \to Q$ as follows:

- $\hat{\delta}(q, \varepsilon) = q$ for all $q \in Q$.
- $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$ for all $q \in Q$, $x \in \Sigma^*$, $a \in \Sigma$.

The language accepted by M is $T(M) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$.
Similarly, for an NFA $M = (Q, \Sigma, \delta, q_0, F)$, we define $\hat{\delta} : Q \times \Sigma^* \to 2^Q$ as follows:

- $\hat{\delta}(q, \varepsilon) = \{q\}$ for all $q \in Q$.
- $\hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q,x)} \delta(p, a)$ for all $q \in Q$, $x \in \Sigma^*$, $a \in \Sigma$.

The language accepted by M is
$T(M) = \{w \in \Sigma^* \mid q_0 \in S_0, \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$.

NFA to DFA

For every NFA $M = (Q, \Sigma, \delta, S_0, F)$, there exists a DFA $M' = (Q', \Sigma, \delta', q_0', F')$ such that $T(M) = T(M')$. The construction is as follows:

- $Q' = 2^Q$
- $q_0' = S_0$
- $\delta' : Q' \times \Sigma \to Q'$ defined as $\delta'(S, a) = \bigcup_{q \in S} \delta(q, a)$ for all $S \in Q'$, $a \in \Sigma$.
- $F' = \{S \in Q' \mid S \cap F \neq \emptyset\}$

Minimize a DFA

For every DFA $M = (Q, \Sigma, \delta, q_0, F)$, there exists a unique (up to isomorphism) minimal DFA $M' = (Q', \Sigma, \delta', q_0', F')$ such that $T(M) = T(M')$ and $|Q'| \leq |Q|$. The construction is as follows:

- Remove unreachable states from Q.
- Define an equivalence relation $\sim$ on Q such that $p \sim q$ if for all $w \in \Sigma^*$, $\hat{\delta}(p, w) \in F$ if and only if $\hat{\delta}(q, w) \in F$.
- The states of M' are the equivalence classes of $\sim$.
- The start state of M' is the equivalence class of q_0.
- The accepting states of M' are the equivalence classes that contain at least one accepting state of M.
- The transition function δ' is defined as $\delta'([q], a) = [\delta(q, a)]$ for all equivalence classes $[q]$ and all $a \in \Sigma$.

Myhill-Nerode Theorem

A language $L \subseteq \Sigma^*$ is regular if and only if the equivalence relation $\equiv_L$ defined as $x \equiv_L y$ if for all $z \in \Sigma^*$, $xz \in L$ if and only if $yz \in L$ has a finite number of equivalence classes.
This theorem is used to construct the minimal DFA for a regular language L. The states of the minimal DFA are the equivalence classes of $\equiv_L$.

Algorithm for Minimizing a DFA

- Create a table with all pairs of states (p, q) where $p, q \in Q$ and $p \neq q$.
- Mark all pairs (p, q) where $p \in F$ and $q \notin F$ or $p \notin F$ and $q \in F$.
- Repeat until no new pairs are marked:
  - For each unmarked pair (p, q), for each $a \in \Sigma$, check the pair $(\delta(p, a), \delta(q, a))$. If this pair is marked, mark (p, q).
- The unmarked pairs represent equivalent states. Merge them to form the states of the minimal DFA.

Pumping Lemma for Regular Languages

If L is a regular language, then there exists a constant p (the pumping length) such that any string $w \in L$ with $|w| \geq p$ can be split into three parts, $w = xyz$, such that:

- $|xy| \leq p$
- $|y| > 0$
- $xy^n z \in L$ for all $n \geq 0$

This can be used to prove that a language is not regular.

Pushdown Automata

A pushdown automaton (PDA) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where

- Q is a finite set of states.
- Σ is the input alphabet. It holds $Q \cap \Sigma = \emptyset$.
- Γ is the stack alphabet. It holds $Q \cap \Gamma = \emptyset$ and $\Sigma \cap \Gamma = \emptyset$.
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$ is the transition function.
- $q_0 \in Q$ is the start state.
- $Z_0 \in \Gamma$ is the initial stack symbol.
- $F \subseteq Q$ is the set of accepting states.

PDA Configurations

A configuration of a PDA is a triple (q, w, γ) where $q \in Q$ is the current state, $w \in \Sigma^*$ is the remaining input, and $\gamma \in \Gamma^*$ is the current stack content (with the top of the stack being the leftmost symbol).
The PDA can make a transition from configuration $(q, aw, \gamma Z)$ to $(p, w, \beta\gamma)$ if $(p, \beta) \in \delta(q, a, Z)$ for $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$, and $\beta \in \Gamma^*$.
The PDA accepts an input string w if there exists a sequence of transitions starting from (q_0, w, Z_0) that leads to a configuration (q, ε, γ) where $q \in F$.

Turing Machines

A Turing machine (TM) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where

- Q is a finite set of states.
- Σ is the input alphabet. It holds $Q \cap \Sigma = \emptyset$.
- Γ is the tape alphabet. It holds $Q \cap \Gamma = \emptyset$ and $\Sigma \subseteq \Gamma$.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function.
- $q_0 \in Q$ is the start state.
- $B \in \Gamma$ is the blank symbol.
- $F \subseteq Q$ is the set of accepting states.

TM Configurations

A configuration of a TM is a triple $(q, w, a\gamma)$ where $q \in Q$ is the current state, $w \in \Gamma^*$ is the content of the tape to the left of the head, and $a\gamma \in \Gamma^*$ is the content of the tape from the head position to the right (with a being the symbol under the head).
The TM can make a transition from configuration $(q, w, a\gamma)$ to $(p, w', b\gamma')$ if $\delta(q, a) = (p, b, D)$ where $D \in \{L, R\}$, and

- If $D = R$, then $w' = wb$ and $\gamma' = \gamma$.
- If $D = L$, then w' is w without its last symbol (if w is not empty) and $\gamma' = cb\gamma$ where c is the last symbol of w (or B if w is empty).

The TM accepts an input string w if there exists a sequence of transitions starting from (q_0, ε, w) that leads to a configuration (q, α, β) where $q \in F$.

Linear Bounded Automata

A linear bounded automaton (LBA) is a Turing machine with the restriction that the tape head cannot move beyond the portion of the tape that contains the input string. In other words, the tape is bounded by the length of the input.
Formally, an LBA is a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where the tape head is not allowed to move left of the leftmost symbol of the input or right of the rightmost symbol of the input.
LBAs recognize context-sensitive languages.

Correspondence between Grammars and Automata

- Regular languages (Type 3) can be recognized by DFAs and NFAs.
- Context-free languages (Type 2) can be recognized by PDAs.
- Context-sensitive languages (Type 1) can be recognized by LBAs.
- Recursively enumerable languages (Type 0) can be recognized by TMs.