1 Prerequisite Definitions

Alphabets Σ, and Γ are finite nonempty sets of symbols.

A *string* is a finite sequence of zero or more symbols from an alphabet.

$\Sigma^\star$ is the set of all strings over alphabet Σ.

ε is the empty string and cannot be in Σ.

A *problem* is a mapping from strings to strings.

A *decision problem* is a problem whose output is yes/no (or often accept/reject).

A decision problem be thought of as the set of all strings for which the function outputs "accept".

A *language* is a set of strings, so any set $S \subseteq \Sigma^\star$ is a language, even $\emptyset$. Thus, decision problems are equivalent to languages.

2 Regular Languages

$L(M)$ is the language accepted by machine M.

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

- Q is a finite set of states,
- Σ is an alphabet,
- $\delta : Q \times \Sigma \to Q$ is a transition function describing its transitions and labels,
- $q_0 \in Q$ is the starting state, and
- $F \subseteq Q$ is a set of accepting states.

If δ is not fully specified, we assume an implicit transition to an *error state*.

A deterministic finite automaton M accepts input string $w = w_1 w_2 \dots w_n$ ($w_i \in \Sigma^\star$) if there exists a sequence of states $r_0, r_1, r_2, \dots, r_n$ ($r_i \in Q$) such that

- $r_0 = q_0$,
- for all $i \in \{1, \dots, n\}$, $r_i = \delta(r_{i-1}, w_i)$, and
- $r_n \in F$.

$r_0, r_1, r_2, \dots, r_n$ are the sequence of states visited during the machine's computation.

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

- Q, Σ, q_0, F are the same as a deterministic finite automaton's, and
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$.

A non-deterministic finite automaton accepts the string $w = w_1 w_2 \dots w_n$ ($w_i \in \Sigma^\star$) if there exist a string $y = y_1 y_2 \dots y_m$ ($y_i \in (\Sigma \cup \{\varepsilon\})^\star$) and a sequence $r = r_0, r_1, \dots, r_n$ ($r_i \in Q$) such that

- $w = y_1 \circ y_2 \circ \cdots \circ y_m$ (i.e. y is w with some ε inserted),
- $r_0 = q_0$,
- for all $i = \{1, \dots, m\}$, $r_i \in \delta(r_{i-1}, q_i)$, and
- $r_m \in F$.

The ε*-closure* for any set $S \subseteq Q$ is denoted $E(S)$, which is the set of all states in Q that can be reachable by following any number of ε-transition.

Theorem 1. *A non-deterministic finite automaton can be converted to an equivalent deterministic finite automaton.*

A *regular language* is any language accepted by some finite automaton. The set of all regular languages is called the *class of regular languages*.

Theorem 2. *Regular languages are closed under*

- *Concatenation* $L_1 \circ L_2 = \{x \circ y : x \in L_1$ *and* $y \in L_2\}$. *Note:* $L_1 \not\subseteq L_1 \circ L_2$.
- *Union* $L_1 \cup L_2 = \{x : x \in L_1$ *or* $x \in L_2\}$.
- *Intersection* $L_1 \cap L_2 = \{x : x \in L_1$ *and* $x \in L_2\}$.
- *Complement* $\overline{L} = \Sigma^\star \setminus L = \{x : x \notin L\}$.
- *Star* $L^\star = \{x_1 \circ x_2 \circ \cdots \circ x_k : x_i \in L$ *and* $k \geq 0\}$.

R is a regular expression if R is

- $a \in \Sigma$,
- ε,
- $\emptyset$,
- $R_1 \cup R_2$, or $R_1 | R_2$,
- $R_1 \circ R_2$, or $R_1 R_2$,
- $R_1^\star$,
- Shorthand: $\Sigma = (a_1 | a_2 | \dots | a_k)$, $a_i \in \Sigma$,

where R_i is a regular expression.

Identities of Regular Languages

- $\emptyset \cup R = R \cup \emptyset = R$
- $\emptyset \circ R = R \circ \emptyset = \emptyset$
- $\varepsilon \circ R = R \circ \varepsilon =$
- $\varepsilon^\star = \varepsilon$
- $\emptyset^\star = \emptyset$
- $\emptyset \cup R \circ R^\star = R \circ R^\star \cup \varepsilon = R^\star$
- $(a|b)^\star = (a^\star|b^\star)^\star = (a^\star b^\star)^\star = (a^\star|b)^\star = (a|b^\star)^\star = a^\star(ba^\star)^\star = b^\star(ab^\star)^\star$

Theorem 3. *Languages accepted by DFAs = languages accepted by NFAs = regular languages*

Theorem 4. *If L is a finite language, L is regular.*

If a computation path of any finite automaton is longer than the number of states it has, there must be a cycle in that computation path.

Lemma 1 (Pumping Lemma)**.** *Every regular language satisfies the pumping condition.*

Pumping condition: There exists an integer p such that for every string $w \in L$, with $|w| \geq p$, there exist strings $x, y, z \in \Sigma^\star$ with $w = xyz, y \neq \varepsilon, |xy| \leq p$ such that for all $i \geq 0$, $xy^iz \in L$.

Negation of pumping condition: For all integers p, there exists a string $w \in L$, with $|w| \geq p$, for all $x, y, z \in \Sigma^\star$ with $w = xyz, y \neq \varepsilon, |xy| \leq p$, there exists $i \geq 0, i \neq 1$ such that $xy^iz \notin L$.

Limitations of finite automata:

- Only read input once, left to right.
- Only finite memory.

3 Context-Free Languages

A pushdown automaton is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where

- Q is a finite set of states,
- Σ is its input alphabet,
- Γ is its stack alphabet,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \to 2^{Q \times (\Gamma \cup \varepsilon)}$ is its transition function,
- $q_0 \in Q$ is its starting state, and
- $F \subseteq Q$ is a finite set of accepting states.

Suppose u, v, w are strings of variables and terminals, and there is a rule $A \to w$. From the string uAv, we can obtain uwv. We write $uAv \to uwv$, and say uAv yields uwv.

If $u_1 \to u_2 \to \cdots \to u_k$, then $u_1 \to^\star u_k$, or u_1 derives u_k. There must be a finite number of arrows between u_1 and u_k.

Given a grammar G, the language derived by the grammar is $L(G) = \{w \in \Sigma^\star : S \to^\star w$ and S is the start variable$\}$

Context-free grammar: the lhs of rules is a single variable, rhs is any string of variables and terminals. A *context-free language* is one that can be derived from a context-free grammar. An example context-free grammar is $G = (V, \Sigma, R, \langle\texttt{EXPR}\rangle)$, where $V = \{\langle\texttt{EXPR}\rangle, \langle\texttt{TERM}\rangle, \langle\texttt{FACTOR}\rangle\}$, $\Sigma = \{a, +, \times, (,)\}$, and $R = \{\langle\texttt{EXPR}\rangle \to \langle\texttt{EXPR}\rangle + \langle\texttt{TERM}\rangle | \langle\texttt{TERM}\rangle, \langle\texttt{TERM}\rangle \to \langle\texttt{TERM}\rangle \times \langle\texttt{FACTOR}\rangle | \langle\texttt{FACTOR}\rangle, \langle\texttt{FACTOR}\rangle \to (\langle\texttt{EXPR}\rangle)\}$.

A *left-most derivation* is a sequence $S \to u_1 \to u_2 \to \cdots \to u_k \to w$ where each step applies a rule to the left-most variable. A grammar is *ambiguous* when it has multiple left-most derivations for the same string.

Theorem 5. *A language L is recognized by a pushdown automaton iff L is described by a context-free grammar.*

Theorem 6. *Context-free languages are closed under union, concatenation, star.*