1 Finite Automata

1.1 Deterministic finate automaton - DFA
is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finate set of *states*,
- Σ is a finite *input alphabet*
- $q_0 \in Q$ is the *initial state*,
- $F \subseteq Q$ is the set of *final states*, and
- δ is the *transition function*, i.e. $\delta : Q \times \Sigma \to Q$
  For each state, there must be a transition for every input symbol out of Σ.

exp. Dfa for finding modulo of binary numbers
Suppose our modulo is m. Then for every possible remainder, there must be a state in fa $\{q_0, q_1, \dots, q_{m-1}\}$.

- state $q_0 : m * k + 0$
  $m * k\ |0 \Rightarrow 2 * (5k) + 0 = m * k + 0$ (on 0, we go to q_0)
  $m * k\ |1 \Rightarrow 2 * (5k) + 1 = m * k + 1$ (on 1, we go to q_1)
- state $q_1 : k + 1$
  $m * k\ |0 \Rightarrow 2 * 1 + 0 = 2$ (go to q_2)
  $m * k\ |1 \Rightarrow 2 * 1 + 1 = 3$ (go to q_3)
- state $q_{m-1} : k + (m - 1)$
  $m * k\ |0 \Rightarrow 2 * (m - 1) + 0$
  $m * k\ |1 \Rightarrow 2 * (m - 1) + 1$

If your remainder is bigger than m, then you must modulo it!

1.2 Nondeterministic finate automaton - NFA
is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q, Σ, q_0, F *read dfa*
- δ is the *transition function*, i.e. $\delta : Q \times \Sigma \to 2^Q$
  That is $\delta(q, a)$ is the *set* of all states p such that there is a transition labeled from a to p.

1.3 NFA with epsilon moves - NFA_ϵ
is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q, Σ, q_0, F *read dfa*
- δ is the *transition function*, i.e. $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$
  That is $\delta(q, a)$ is the *set* of all states p such that there is a transition labeled from a to p, *where **a** is either a symbol in Σ or ϵ.*

ϵ-closure defines which ϵ transitions are allowed from a single state in a fa (set of states we can reach).

exp. NFA for L^c

$$NFA(L) \to DFA(L) \to DFA(L^c)$$

Due to the properties of **DFA**, the complementation is applied just by switching final and non-final states of fa.

2 Regular expressions

2.1 Regular operations
Let L_1, L_2 be some regular languages. Then their

- **union** $\to L_1 \cup L_2 = \{\forall x : x \in L_1 \text{ or } x \in L_2\}$
- **concatenation** $\to L_1.L_2 = L_1L_2$
- **kleene closure** $\to= L^*$

are also regular languages.
Regexp are equvalent with NFA.

2.2 Pumping lemma for regular languages Let R be a class of regular languages. Then language $L \in R \to \exists n > 0$:

$$\forall z \in L, |z| \geq n :$$
$$\exists u, v, w : \ |uv| \leq n, |v| \geq 1, z = uvw \to \forall i \geq 0 : uv^iw \in L$$

if we negate lemma, we can prove that some languages are irregular

$$\forall n > 0 : \exists z \in L, |z| \geq n$$
$$\forall u, v, w : \ |uv| \leq n, |v| \geq 1, z = uvw \to \exists i \geq 0 : uv^iw \notin L \Rightarrow L \notin R$$

3 Context-free grammars

3.1 Definition: A context-free grammar (CFG)
is a 4-tuple $G = (V, T, P, S)$ where:

- V is a finite set of *variables*
- T is a finite set of *terminals*
- P is a finite set of *productions*
  each of which is of the form $A \to \alpha$,
  where $A \in V$ and α is a word in the language $(V \cup T)^*$
- S is a special variable called the *start symbol*

Ambiguity: A CFG is said to be ambiguous if some word has more than one derivation tree.

exp. regex to CFG conversion
Suppose we have a regex: $a(ab)^*bb(aa + b)^*a$
Then we could model a CFG for it as:

- $S \to XYZUV$
- $X \to a$
- $Y \to aabY|\epsilon$
- $Z \to bb$
- $U \to aaU|bU|\epsilon$
- $V \to a$

3.2 Pumping lemma for context-free languages Let L be a CFL.

$$\exists n > 0 :$$
$$\forall z \in L, |z| \geq n :$$
$$\exists u, v, w, x, y : \ |vwx| \leq n, |vx| \geq 1$$
$$z = uvwxy \to \forall i \geq 0 : uv^iwx^iy \in L$$

if we negate lemma, we can prove that some languages are not context-free.

$$\forall n > 0 :$$
$$\exists z \in L, |z| \geq n :$$
$$\forall u, v, w, x, y : \ |vwx| \leq n, |vx| \geq 1$$
$$z = uvwxy \to \exists i \geq 0 : uv^iwx^iy \notin L$$

4 Pushdown Automata

4.1 Definition: 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*
- Γ is the *stack alphabet*
- $q_0 \in Q$ is the *initial state*
- $Z_0 \in \Gamma$ is the *start stack symbol*
- $F \subseteq Q$ is the set of *final states*, and
- δ is the *transition function*
  i.e. a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$ $\to 2^{Q \times \Gamma^*}$

4.2 Accepted languages of the PDA
For PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ we define two languages:

- $L(M)$, the *language accepted by final state*, to be
  $L(M) = \{w \in \Sigma^* | (q_0, w, Z_0) \rightarrow^* (p, \epsilon, \gamma)$
  $for\ some\ p\ \in F\ and\ \gamma \in \Gamma^*\}$

- $L(M)$, the *language accepted by empty stack*, to be
  $N(M) = \{w \in \Sigma^* | (q_0, w, Z_0) \rightarrow^* (p, \epsilon, \epsilon);\ for\ some\ p\ \in Q\}$

5 Touring Machines

5.1 Definition: A simple Toruing 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*
- Γ is the *tape alphabet* $B \in \Gamma => \Sigma \subseteq \Gamma$
- δ is the *transition function*
- q_0 is the *initial state* and,
- $F \subseteq Q$: is the set of *final states*