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** $\to L_1.L_2 = L_1 L_2$
- **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 $L \in R \to \exists n > 0$:

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

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

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

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:

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

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

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

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

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

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 basic 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*

5.2 TM modifications:

- Finite storage $\Rightarrow \delta : Q \times \Gamma \times \Gamma^k \rightarrow Q \times \Gamma \times \{L, R, S\} \times \Gamma^k$
- Multiple track tape $\Rightarrow \delta : Q \times \Gamma^{tk} \rightarrow Q \times \Gamma^{tk} \times \{L, R, S\}$
- Two-way infinite tape $\Rightarrow \delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$
- Multiple tapes $\Rightarrow \delta : Q \times \Gamma^{tp} \rightarrow Q \times (\Gamma \times \{L, R, S\})^{tp}$
- Multidimensional tape $\Rightarrow \delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L_1, R_1, \ldots, L_d, R_d, S\}$

5.3 Universal Touring Machine (UTM) is a TM, that accepts some *Touring machine* **M** description and a *word* **w**. The universal TM then decides if $w \in L(M)$.

[TM description|w] $111 < q_1 > 11 < q_2 > 11 \ldots 11 < q_k > 111w$

$\rightarrow$ language L is **semi-decidable**, if there exists a TM, which:
for every $w \in L$, TM halts in a final state

$\rightarrow$ language L is **decidable**, if there exists a TM, which:
for every $w \in L$, TM halts in a final state
for every $w \notin L$, TM halts in a non-final state

$\rightarrow$ language L is **undecidable**, if it is not decidable.

5.4 Theorems for sets

Let S, A, B be arbitrary sets. Then:

- S is *decidable* $\Rightarrow$ S is *semi-decidable*
- S is *decidable* $\Rightarrow$ $\overline{S}$ is *decidable*
- S and $\overline{S}$ are *semi-decidable* $\Rightarrow$ S is *decidable*
- A and B are *semi-decidable* $\Rightarrow$ $A \cap B$ & $A \cup B$ are *semi-decidable*
- A and B are *decidable* $\Rightarrow$ $A \cap B$ & $A \cup B$ are *decidable*

5.5 Three possibilities for set complementation:

- S and $\overline{S}$ are *decidable*
- S and $\overline{S}$ are *undecidable*, one is semi, and the other is not.
- S and $\overline{S}$ are *undecidable*, and neither is semi-decidable.

5.6 Known languages:

- *Diagonalizable language* $\rightarrow L_d = \{< M > | < M > \notin L(M)\}$ - undecidable / not semi-decidable.
- *Universal language* $\rightarrow L_u = \{(< M >, w) | w \in L(M)\}$ - semi-decidable, but not decidable.
- *Empty language* $\rightarrow L_e\{< M > | L(M) = \{\}\}$ - undecidable
- *Non-Empty language* $\rightarrow L_{ne} = \{< M > | L(M) \neq \{\}\}$ - semi-decidable, but not decidable.

5.6 Rice's theorem for (not)semi-decidabilty:

1. $L \in S \wedge L \subseteq L' \Rightarrow L' \in S$
2. $L \in S \wedge L\ infinite \Rightarrow \exists L' \subseteq L : L \in S,\ L'\ finite$
3. innumerability of final sets in S

$$(1) \wedge (2) \wedge (3) \Leftrightarrow L_s \text{is semi-decidable}$$

6 Complexity classes

6.1 In terms of formal languages:

- DTIME($T(n)$) = $\{L|$ L is a language $\wedge$ L has time complexity T(n)$\}$
- DSPACE($S(n)$) = $\{L|$ L is a language $\wedge$ L has space complexity S(n)$\}$
- NTIME($T(n)$) = $\{L|$ L is a language $\wedge$ L has nondet. time complexity T(n)$\}$
- DSPACE($S(n)$) = $\{L|$ L is a language $\wedge$ L has nondet. space complexity S(n)$\}$

6.2 In terms of decision problems:

- DTIME($T(n)$) = $\{D|$ D is a decision problem $\wedge$ L(D) has time complexity T(n)$\}$
- DSPACE($S(n)$) = $\{D|$ D is a decision problem $\wedge$ L(D) has space complexity S(n)$\}$
- NTIME($T(n)$) = $\{D|$ D is a decision problem $\wedge$ L(D) has nondet. time complexity T(n)$\}$
- DSPACE($S(n)$) = $\{D|$ D is a decision problem $\wedge$ L(D) has nondet. space complexity S(n)$\}$

6.3 Relations between different complexity classes:

- DTIME($T(n)$) $\subseteq$ DSPACE($T(n)$) i.e. What can be solved in time $O(T(n))$, can also be solved on space $O(T(n))$
- $L \in$ DSPACE($S(n)$) $\wedge S(n) \geq log_2 n \Rightarrow \exists c : L \in DTIME(c^{S(n)})$ i.e. What can be solved nondeterminstically in space $O(S(n))$, can be solved deterministically in (at most) time $O(c^{S(n)})$
- $L \in NTIME(T(n)) \Rightarrow \exists c : L \in DTIME(c^{T(n)})$ i.e What can be solved nondeterminstically in time $O(T(n))$, can be solved deterministically in (at most) time $O(c^{T(n)})$
  Consequently, the substitution of nondeterministic algorithm with a deterministic one causes at most *exponential* increase in the **time** required to (deterministically) solve a problem.
- $NSPACE(S(n)) \subseteq DSPACE(S^2(n))$, if $S(n) \geq log_2 n \wedge S(n)$ is *"well behaved"* i.e What can be solved nondeterminstically on space $O(S(n))$, can also be solved deterministically on space $O(S^2(n))$
  Consequently, the substitution of nondeterministic algorithm with a deterministic one causes at most *quadratic* increase in the **space** required to (deterministically) solve a problem.

6.4 Define P, NP, PSPACE, NPSPACE:

- $P = \cup_{i \geq 1}$ DTIME(n^i) is the class of all decision problems **deterministically** solvable in *polynomial time*.
- $NP = \cup_{i \geq 1}$ NTIME(n^i) is the class of all decision problems **nondeterministically** solvable in *polynomial time*.
- $PSPACE = \cup_{i \geq 1}$ DSPACE(n^i) is the class of all decision problems **deterministically** solvable on *polynomial space*.
- $NPSPACE = \cup_{i \geq 1}$ NSPACE(n^i) is the class of all decision problems **nondeterministically** solvable on *polynomial space*.

6.5 Relations between P, NP, PSPACE, NPSPACE:

$$P \subseteq NP \subseteq PSPACE = NPSPACE$$

Proof:

- $P \subseteq NP \rightarrow$ Every deterministic TM of polynomial time complexity can be viewed as a (trivial) nondeterministic TM of the same complexity.
- NP $\subseteq$ PSPACE $\rightarrow$ If $L \in NP$, then $\exists k$ such that $L \in NTIME(n^k)$. So $L \in$ NSPACE(n^k), and hence $L \in$ DSPACE(n^{2k}). Therefore $L \in$ PSPACE.
- (PSPACE = NPSPACE) $\rightarrow$ Trivially, PSPACE $\subseteq$ NPSPACE. The opposite direction: NPSPACE = (def) = $\cup$ NSPACE(n^i) $\subseteq$ (by Savitch) $\subseteq \cup$ DSPACE(n^j) $\subseteq$ PSPACE

6.6 NP-complete & NP-hard problems

NP-hard: $D \leq^p D^*$, for every $D \in NP$.
NP-complete: $D^* \in NP \wedge D \leq^p D^*$, for every $D \in NP$.
Hence, D^* is NP-complete if D^* is in NP and D^* is NP-hard.