1. Automata and Languages

1.1. Regular Languages

Def. A (deterministic) **finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_o, F)$, where

1. Q is a finite set called the **states**,
2. Σ is a finite set called the **alphabet**,
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**,
4. $q_o \in Q$ is the **start state**, and
5. $F \subseteq Q$ is the set of **accepted/final states**

Def. A language is called a **regular language** if some finite automaton recognizes it.
Ex. A language that has strings ending with 0; A language that has strings with substring 010.

The class of regular languages is closed under the following operations:

1. **Union**: $A \cup B = \{x \mid x \in A or x \in B\}$.
2. **Concatenation**: $A \circ B = \{xy \mid x, y \in A, B\}$.
3. **Star**: $A^* = \{x_1, x_2, \ldots, x_k \mid k \geq 0$ and each $x_i \in A\}$.

Determinism- When the machine is in a given state and reads the next input symbol, the next state is unique and already determined.
Nondeterminism- Several choices may exist for the next state at any point.

Def. A **nondeterministic finite automaton** is the same 5-tuple, except

$$\delta : Q \times \Sigma_\epsilon \to P(Q)$$

Theorem. Every NFA has an equivalent DFA.
Corollary. A language is regular if and only if some NFA recognizes it.

Def. R is a **regular expression** if R is

1. a for some a is the alphabet Σ,
2. ε,
3. ϕ,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

Theorem. A language is regular iff some regular expression describes it.
Note. Every regular language can be converted into an NFA.
Note. DFAs can be reduced to minimized DFAs.

Nonregular languages are those that cannot be recognized by DFAs.
Ex. $\{0^n 1^n | n \geq 0\}$.

Theorem. Pumping Lemma If A is a regular language, then $\exists$ a number p (the pumping length) where if any $s \in A$ s.t. $|a| \geq p$, then s can be divided into 3 pieces, $s = xyz$, s.t.

1. for each $i \geq 0, xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Pumping Lemma can help differentiate between regular and nonregular languages.

1.2. Context-free grammars

Def. A CFG is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the **variables**,
2. Σ is a finite set, disjoint from V , called the **terminals**,
3. R is a finite set of **rules**, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the **start variable**.

A left hand derivation exists for every string $s \in L(CFG)$.
The language of the grammar is $\{w \in \Sigma^* | S \Rightarrow^* w\}$.
Grammars can be unambiguous (i.e. each string has a unique LH derivation) or ambiguous (i.e. string has two or more LH derivations), or even inherently ambiguous.
Note. A context-free grammar is in Chomsky normal form if every rule is of the form

$$A \to BC$$
$$A \to a$$
$$S \to \varepsilon$$

Def. A (nondeterministic) **pushdown automaton** is a 6-tuple $(Q, \Sigma, \varsigma, \delta, q_0, F)$, where

1. Q is the set of **states**,
2. Σ is the **input alphabet**,
3. ς is the **stack alphabet**,
4. $\delta : Q \times \Sigma_\varepsilon \times \varsigma_\varepsilon \Rightarrow P(Q \times \varsigma_\varepsilon)$ is the **transition function**,
5. $q_0 \in Q$ is the **start state**, and

6. $F \subseteq Q$ is the set of **accept states**.

The stack gives the PDA a power of small storage.
Theorem.A language A is a CFL iff $\exists$ a PDA P s.t. $L(P) = A$.
Ex. $\{0^i1^j2^k | i \neq j or j \neq k, i, j, k \geq 0\} \in CFL$.

Def. A **deterministic PDA** is the same 5-tuple, except

$$\delta : Q \times \Sigma_\varepsilon \times \varsigma_\varepsilon \Rightarrow Q \times \varsigma_\varepsilon \cup \{\phi\}.$$

s.t. $\forall q \in Q, a \in \Sigma, b \in \varsigma$ we have exactly one of the following to be non-empty:

$$\delta(q, a, b), \delta(q, a, \varepsilon), \delta(a, \varepsilon, b), \delta(\varepsilon, \varepsilon, \varepsilon)$$

Theorem.A language A is a DCFL iff $\exists$ a DPDA D s.t. $L(D) = A$.

Ex. $\{0^n1^n | n \geq 0\} \in DCFL \setminus RL$.

Note. PDAs can either accept by final state or by emptying the stack.
Note. Every DPDA has an equivalent DPDA that always reads the entire input string.
Note. If $A = N(P)$,i.e. accepted by emptying the stack, then A is prefix-free.

Note. CFLs are closed under union, intersections and complement.
Note. DCFLs are closed under complement.

Def. A **DCFG** is a CFG such that every valid string has a forced handle. $\exists$ a similar pumping lemma for non-context-free languages where the string can be divided into five pieces instead, $s = uv^ixy^iz$.

2. Computability Theory

Def. A **Turing Machine** is a 7-tuple $(Q, \Sigma, \tau, \delta, q_o, q_a, q_r)$ where

1. Q is a finite set called the **states**,
2. Σ is a finite set called the **alphabet** and $B \notin \Sigma$,
3. τ is the **tape alphabet** where $B \in \tau$ and $\Sigma \subseteq \tau$
4. $\delta : Q \times \tau \to Q \times \tau \times \{L, R\}$ is the **transition function**,
5. $q_o \in Q$ is the **start state**,
6. $q_a \in Q$ is the **accept state**, and
7. $q_r \in Q$ is the **reject state**.

Def. The **configuration** of a machine is of the form $u_1u_2 \dots u_{i-1}qu_i \dots u_n$ where $u_j \in \tau$. We say the machine is at state q and pointing to u_i.

1. **Start:** $q_0u_1u_2 \dots u_n$
2. **Accept:** $u_1u_2 \dots u_{i-1}q_au_i \dots u_n$.
3. **Reject:** $u_1u_2 \dots u_{i-1}q_ru_i \dots u_n$.

Note. During it's running, at any point if TM enters q_{accept} or q_{reject}, it breaks and accepts or rejects respectively. This is known as accepting/rejecting by halting. If TM does not halt, a string is rejected by looping.

Church-Turing Thesis. Turing machines capture all algorithms, i.e. existence of an algorithm $\Rightarrow \exists$ a turing machine that can run it.

Def. A Language A is a **turing recognizable language** if $\exists$ a TM M such that $L(M) = A$.
Def. A Turing Machine M is a **Decider** if M halts for all input strings.
Def. A Language A is a **turing decidable language** if $\exists$ a decider M such that $L(M) = A$.
Note. Turing Decidable $\Rightarrow$ Turing Recognizable.
Theorem. All CFLs are decidable Turing Languages.

Note. $A_{TM} = \{ < M, w > |$ M accepts w $\}$ is Turing recognizable but not Turing decidable, i.e., it is undecidable.
Def. A language A is called **Co-Turing recognizable** if A^c is Turing recognizable.
Theorem. A language A is decidable $\Leftrightarrow$ A is Turing Recognizable and Co-Turing Recognizable.
Corollary. $\overline{A}_{TM}$ is Turing Unrecognizable.

Def. For two languages A and B, **A reduces to B** means that $\exists$ a decider for B $\Rightarrow \exists$ a decider for A.
Note. If A reduces to B and B is decidable $\Rightarrow$ A is decidable.
Note. If A reduces to B and A is undecidable $\Rightarrow$ B is undecidable.
Ex. $A_{HALT} = \{ < M, w > |$ M halts on w $\}$. A_{TM} reduces to A_{HALT} and A_{TM} is undecidable $\Rightarrow A_{HALT}$ is undecidable.

Def. A function f : $\Sigma^* \to \Sigma^*$ is a **computable function** if $\exists$ a TM M such that $\forall w \in \Sigma^*$, M halts with tape content as f(w).
Def. A language A is **Mapping Reducible** to language B (A $\leqslant_M$ B) if $\exists$ a computable function f : $\Sigma^* \to \Sigma^*$ such that $\forall w \in \Sigma^*$ and $w \in A \Leftrightarrow f(w) \in B$.
Theorem. A $\leqslant_M$ B and B is decidable $\Rightarrow$ A is decidable.
Theorem. A $\leqslant_M$ B and A is undecidable $\Rightarrow$ B is undecidable.

3. Complexity Theory

Def. The **running time** of Turing Machine M is the function $f : N \Rightarrow N$, where $f(n)$ is the max number of steps that M uses on any input of length n.
Note. We say $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0, f(n) \leq c \cdot g(n)$.
Def. The time complexity class, $TIME(t(n))$, is the collection of all languages that are decidable by an $O(t(n))$ time Turing Machine.
Note. All reasonable deterministic computational models are polynomially equivalent, i.e., any one of them can simulate another with only a polynomial increase in running time.

Def. P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine,

$$P = \bigcup_k TIME(n^k).$$

Ex. $PATH = \{< G, s, t > |G$ is a directed graph that has a directed path from s to t$\}$; $RELPRIME = \{< x, y > |x$ and y are relatively prime$\}$.
Theorem. Every CFL is a member of P.

Def. A **verifier** for a language A is an algorithm V, where

$$A = \{w|V \text{ accepts } < w, c > \text{ for some string } c\}.$$

Here, c is additional information, also called a **certificate.**

A **polynomial time verifier** runs in polynomial time in the length of w.
Theorem. NP is the class of languages that have polynomial time verifiers.
Ex. $HAMPATH = \{< G, s, t >| \ G$ is a directed graph with a Hamiltonian path from s to $t\}$.
Theorem. A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.
Def.

$$NTIME(t(n)) = \{L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine}\}.$$

The P vs. NP Problem.
P = the class of languages for which membership can be decided quickly.
NP = the class of languages for which membership can be verified quickly.

$$A \in P \Rightarrow A \in NP$$

Def. Language A is **polynomial time reducible** to language B ($A \leqslant_P B$), if $\exists$ a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ s.t.

$$w \in A iff f(w) \in B \forall w \in A$$

Note. B is called $NP-hard$ if $A \leqslant_P B \forall A \in NP$.
Note. B is called $NP-complete$ if B is $NP-hard$ and $B \in NP$.
Ex. $SAT, KSAT \in NP-complete$.