1 Chapter One

1.1 Automaton

Def 1.1 (Finite Automaton)**.** A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. δ: $Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

Def 1.2 (Nondeterministic Finite Automaton)**.** A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. δ: $Q \times \Sigma \longrightarrow \mathscr{P}(Q)$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

Def 1.3 (Generalized Nondeterministic Finite Automaton)**.** A ***generalized nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:

1. Q is a finite set called the ***states***,
2. Σ is the ***input alphabet***,
3. $\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \longrightarrow \mathcal{R}$ is the ***transition function***,
4. q_{start} is the ***start state***, and
5. q_{accept} is the ***accept state***.

The symbol $\mathcal{R}$ is the collection of all regular expressions over the alphabet Σ, and q_{start} and q_{accept} are the start and accept states. If $\delta(q_i, q_j) = R$, the arrow from state q_i to state q_j has the regular expression R as its label. The domain of the transitition function is $(Q - \{q_{accept}\}) \times (Q - \{q_{start}\})$ because an arrow connects every state to every other state, except that no arrows are coming from q_{accept} or going to q_{start}.

1.2 Computation

Def 1.4 (Computation)**.** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \dots w_n$ be a string where each w_i is a member of the alphabet Σ. Then M accepts w if a sequence of states $r_0, r_1, \dots, r_n \in Q$ exists with three conditions:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \dots, n-1$, and
3. $r_n \in F$.

Def 1.5 (recognizes)**.** M ***recognizes language*** A if $A = \{w : M \text{ accepts } w\}$.

1.3 Regular

Def 1.6 (Regular)**.** A language is called a ***regular language*** if some finite automaton recognizes it.

Def 1.7 (Regular Operations)**.** Let A and B be languages. We define the regular operations ***union***, ***concatenation*** and ***star*** as follows:

- **Union**: $A \cup B = \{x : x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy : x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1 x_2 \dots x_k : k \geq 0 \text{ and each } x_i \in A\}$

In arithmetic, we say that $\times$ has precedence over over $+$ to mean that when there is a choice, we do the $\times$ operation first. Thus in $2 + 3 \times 4$, the 3×4 is done before the addition. To have the addition done first, we must add paretheses to to obtain $(2 + 3) \times 4$. In regular expressions, the star operation is done first, followed by concatenation, and finally union, unless parentheses change the usual order.

Def 1.8 (Regular Operations)**.** Say that R is a *expression* if R is:

1. a for some A in the alphabet Σ,
2. ε,
3. $\varnothing$,
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.

In items 1 and 2, the regular expressions A and ε represent the languages $\{a\}$ and $\{\varepsilon\}$, respectively. In item 3, the regular expression $\varnothing$ represents the empty language. In items 4, 5, and 6, the expressions represent the languages obtained by taking the union or concatenation of the languages R_1 and R_2, or the star of the language R_1, respectively.

Def 1.9 (Equivalence With Finite Automata)**.** A language is ***regular*** if and only if some regular expression describes it.

Def 1.10. Equivalence With Regular Expressions **Def 1.9** means as well that if a language is described by a ***regular expression***, then it is ***regular***.

1.4 Nonregular Languages

Def 1.11 (Puming Lemma)**.** If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

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

2 Chapter Two

2.1 Context-Free Grammar

Def 2.1 (Context-Free Grammar)**.** A ***context-free grammar*** is a 4-tuple (V, Σ, R, S), where :—

1. V is a finite set called the ***variable***,
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.

The ***language of the grammar*** is $\{w \in \Sigma^* : S \overset{*}{\Rightarrow} w\}$. That is, say that u ***derives*** v, written $u \overset{*}{\Rightarrow} v$, if $u = v$ or if a sequence $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ and

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v.$$

Def 2.2 (Chomsky Normal Form)**.** A context-free grammar is in Chomsky normal form if every rule is of the form :—

$$A \to BC$$
$$A \to a$$

where a is any terminal and A, B, and C are any variable—except that B and C may not be the start variable. In addition, we permit the Rule $S \to \varepsilon$, where S is the start variable.

2.2 Pushdown Automata

Def 2.3 (Pushdown Automaton)**.** A ***pusdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and :—

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

Recall that $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$. The domain of the transition function is $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$. Thus the current state, next input symbol read, and top symbol of the stack determine the next move of a pushdown automaton.

Def 2.4 (Equivalence With Context-Free Grammar)**.** A language is context free if and only if some pushdown automaton recognizes it. If a language is context free, then some pushdown automaton recognizes it.

3 Chapter Three

3.1 Turing Machine

Def 3.1 (Turing Machine)**.** A ***Turing machine*** is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accepts}, q_{reject})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the blank symbol _,
3. Γ is the tap alphabet, where _ $\in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L,R}\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

Def 3.2 (Multitape Turing Machine)**.** The only difference is that

$$\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}^k,$$

where k is the number of tapes. The expression

$$\delta(q_i, a_1..., a_k) = (q_j, b_1, ..., b_k, \mathrm{L}, \mathrm{R}, ..., \mathrm{L})$$

means that if the machine is in state q_i and heads 1 through k are reading symbols a_1 through a_k, the machine goes to state q_j, writes symbols b_1 through b_k, and directs each head to mave left or right, or to stay put, as specified.

Def 3.3 (Nondeterministic Turing Machine)**.** The only difference is that

$$\delta : Q \times \Gamma \longrightarrow \mathscr{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}).$$

All of the nondeterministic paths can be simulated on a multitape Turing Machine.

- ***start configuration***
- ***accepting configuration***
- ***rejecting configuration***
- ***halting configuration***

- Call a language ***Turing-recognizable*** if some Turing machine recognizes it. (***recursively enumerable language***)
  - A language is Turing-recognizable $\iff$ some enumerator enumerates it.
- Call a language ***Turing-decidable*** or simply decidable if some Turing machine decides it. (***recursive language***)
  - A language is decidable $\iff$ some nondeterministic Turing machine decides it.

4 Chapter Four

4.1 Decidable

- $A_{DFA} = \{\langle B, w\rangle : B \text{ is a DFA that accepts input string } w\}$ is a decidable language.
  - $M =$ "On input B, w:
    1. Simulate B on input w.
    2. If the simulation ends in an accept state, *accept*. If it ends in a nonaccepting state, *reject*"

- $A_{NFA} = \{\langle B, w\rangle : B \text{ is an NFA that accepts input string } w\}$ is a decidable language.
  - N = "On input $\langle B, w\rangle$:
    1. Convert NFA B to an equivalent DFA C, using the procedure for this conversion given in Theorem 1.39.
    2. Run TM M from Theorem 4.1 on input $\langle C, w\rangle$.
    3. If M accepts, *accept*; otherwise, *reject*."
- $A_{REX} = \{\langle R, w\rangle : R \text{ is a regular expression that generates string } w\}$ is a decidable language.
  - P = "On input $\langle R, w\rangle$:
    1. Convert regular expression R to an equivalent NFA A be using the procedure for this conversion given in Theorem 1.54.
    2. Run TM N on input $\langle A, w\rangle$.
    3. If N accepts, *accept*; if N rejects, *reject*."
- $E_{DFA} = \{\langle A\rangle : A \text{ is a DFA and } L(A) = \varnothing\}$ is a decidable language.
  - T = "On input $\langle A\rangle$:
    1. Mark the start state of A.
    2. Repeat until no new states get marked:
       (a) Mark any state that has a transition coming into it from any state that is already marked.
    3. If no accept state is marked, *accept*; otherwise, *reject*."
- $EQ_{DFA} = \{\langle A, B\rangle : A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ is a decidable language.
  - This new DFA C accepts only those strings that are accepted by either A or B but not by both. Thus, if A and B recognize the same language, C will accept nothing.
    $$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right)$$
  - F = "On input $\langle A, B\rangle$:
    1. Construct DFA C as described.
    2. Run TM T from Theorem 4.4 on input $\langle C\rangle$.
    3. If T accepts, *accept*. If T rejects, *reject*."
- $A_{CFG} = \{\langle G, w\rangle : G \text{ is a CFG that generates string } w\}$ is a decidable language.
- $E_{CFG} = \{\langle G\rangle : G \text{ is a CFG and } L(G) = \varnothing\}$ is a decidable language.

4.2 Undecidable

- $A_{TM} = \{\langle M, w\rangle : M \text{ is a TM and } M \text{ accepts } w\}$ is an undecidable language. It is also Turing-recognizable. recognizers *are* more powerful than deciders. U recognizes A_{TM}, it is the universal Turing machine capable of simulating every other Turing machine.
  - U = "On input $\langle M, w\rangle$:
    1. Simulate M on input w.
    2. If M ever enters its accept state, *accept*; if M ever enters its reject state, *rejects*."
  - The proof of this relies on a technique called *diagonalization*.
- $HALT_{TM} = \{\langle M, w\rangle : M \text{ is a TM and } M \text{ accepts } w\}$ is an undecidable language. It is also Turing-recognizable. Let's assume for the purpose of obtaining a contradiction that TM R decides $/\text{HALT}/_{\text{TM}}$
  - U = "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:
    1. Run TM R on input $\langle M, w\rangle$.
    2. If R rejects, *reject*.
    3. If R accepts, simulate M on w until it halts.
    4. If M has accepted, *accept*; if M has rejected, *reject*."
  - Clearly, if R decides HALT_{TM}, then S decides A_{TM}. Because A_{TM}. Because A_{TM} is undecidable, HALT_{TM} also must be undecidable.

5 Chapter Five

5.1 Reducible

Def 5.1 (Computation History)**.** Let M be a Turing Machine and w an input string. An **accepting computation history** for M on w is a sequence of configurations, $C_1, C_2, ..., C_l$, where C_1 is the start configuration of M on w, C_l is an accepting configuration of M on w, C_l is an accepting configuration of M, and each C_i legally follows from C_{i-1}according to the rules of M. A **rejecting computation history** for M on w is defined similarly, except that C_l is a rejecting configuration.

Def 5.2 (Linear Bounded Automaton)**.** A **linear bounded automaton** is a restricted type of Turing machine wherein the tape head isn't permitted to move off the portion of the tape containing the input. If the machine tries to move its head off either end of the input, the head stays where it is—in the same way that the head will not move off the left-hand end of an ordinary Turing machine's tape.

Let M be an LBA with q states and g symbols in the tape alphabet. There are exactly qng^n distinct configurations of M for a tape of length n.

- A_{LBA} is decidable. E_{LBA} is undecidable. $E_{LBA} = \{\langle M\rangle \mid M \text{ is an LBA and } L(M) = \varnothing\}$

- ALL_{CFG} is undecidable. $ALL_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$

Def 5.3 (Computable Function)**.** A function $f : \Sigma^* \longrightarrow \Sigma^*$ is a **computable function** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Def 5.4 (Mapping Reducible, Reduction)**.** Language A is **mapping reducible** to language B, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \longrightarrow \Sigma^*$, where for every w,

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

The function f is called the **reduction** from A to B.