Theory of Computation

September 24, 2022

Contents

1 Regular languages

1.1 DFA

A deterministic finite automatum (DFA) is represented by the tuple $(Q, \Sigma, \delta, q_0, F)$.

Some languages accepted by DFA:

1. $\{w \in \{0,1\}^* \mid w$ has even number of 0's$\}$.
2. $\{w \in \{a,b,c\}^* \mid w$ ends in $ab\}$.
3. $\{w \in \{0,1\}^* \mid w$ is divisible by $3\}$.
4. $\{w \in \{0,1\}^* \mid$ 3rd last symbol of w is a $1\}$. This can be generalized.
5. $\{x \mid |x| = 3\}$. This can be generalized.
6. $\{w \in \{0,1\}^* \mid w$ does not contain 11 as a substring$\}$.

For the last language, use dump state to remove substrings 11. A dump state is a state from where the automaton cannot reach an accept state.

For a language, there can be multiple DFAs accepting it. But every DFA accepts exactly one language.

The state space Q partitions the set of all strings.

Let $L \subset \Sigma^*$. We say that L is regular if there exists a DFA M such that $L = L(M)$.

A non-regular language is $\{0^n 1^n \mid n > 0\}$. This involves counting and keeping track of the count.

Let $A, B \in \Sigma^*$. The following are called regular operations.

1. Union: $A \cup B = \{x \mid x \in A$ or $x \in B\}$.
2. Concatenation: $AB = A \cdot B = \{xy \mid x \in A$ and $y \in B\}$.
3. Kleene star: $A^* = \{x_1 \cdots x_k \mid k \geq 0$ and $x_i \in A$ for all $i\}$.

Note that $\epsilon \in A^*$. This is the empty string.

1.2 NFA

In a non-deterministic finite automatum (NFA), a state may jump to multiple states simultaneously. Computation occurs simultaneously along each path.

An input is accepted if there is some computation path leading to accept state.

1. ϵ-transitions: moving from one state to another without any input.
2. The transitions are labelled with symbols from the set $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$.

The description of NFA is almost like that of DFA except that the transition function looks like $\delta : Q \times \Sigma_\epsilon \to 2^Q$.

As in DFA, we define $L(N) = \{w \in \Sigma^* \mid N \text{ accepts } w\}$, where N is an NFA.

Examples of languages accepted by NFA:

1. $\{w \mid |w| \text{ is divisible by 2 or 5}\}$. Bifurcate the cases using ϵ-transitions.

By default, every DFA is an NFA. But any NFA N has a corresponding DFA D such that $L(N) = L(D)$.

States that are not reachable from the start state are called "dead states".

Closure property: regular languages are closed under union, concatenation and Kleene star operations.

1.3 RegEx

Regular expressions (RegEx) are algebraic representation of languages. They find applications in text editors, compiler design, etc.

The union $\cup$ is often written as $+$ and the concatenation $\cdot$ is dropped.

RegEx	Language
0	$\{0\}$
ϵ	$\{\epsilon\}$
$0 + 01$	$\{0, 01\}$
$(0 + 01)1^*$	$\{0, 01, 01, 011, \ldots\}$

For every RegEx RE, there is a unique language $L(RE)$. The converse may not be true.

Precedence of the operators (high to low): $(), *, \cdot, \cup$.

Also, $\emptyset^* = \{\epsilon\}$.

Some more examples:

Language	RegEx
$\{w \mid w \text{ has a single } 1\}$	0^*10^*
$\{w \mid w \text{ has at most a single } 1\}$	$0^* + 0^*10^*$
$\{w \mid \lvert w\rvert \text{ is divisible by } 3\}$	$((0+1)(0+1)(0+1))^*$

Union $+$ and concatenation $\cdot$ are associative operations on RegEx. Also, $\cdot$ distributes over $+$ both from the left and right. $+$ commutes. Also, $R + \emptyset = R$ and $R \cdot \epsilon = R$. Finally, $(R^*)^* = R^*$.

A language L is regular iff there exists a RegEx R such that $L = L(R)$.

1.4 GNFA

A generalized non-deterministic finite automatum (GNFA) is just like an NFA except that it has RegEx labelling its transitions.

GNFA is quite useful in converting a DFA into RegEx. We modify the given DFA into a GNFA so that the following hold:

1. There is a unique accept state.
2. There are no incoming transitions to the start state and no outgoing transitions from the accept state.
3. There are transitions from start state to every other state and from every state to the accept state.
4. There is a transition between every pair of states that are not the start/accept state.

Remove each state one at a time without messing up transitions. You should end up with a RegEx and just the start state and accept state.

1.5 Closure

Complement of a language L is defined by $\bar{L} = \Sigma^* - L$.

Regular languages are closed under complementation. Simply change the set of final states to $F' = Q - F$.

Regular languages are also closed under intersection. This follows from De Morgan's law and closure under complementation and union.

Regular languages are also under closed set difference.

For a language $L \subset \Sigma^*$, we define $\text{rev}(L) = \{w \in \Sigma^* \mid \text{rev}(w) \in L\}$. If L is regular $\text{rev}(L)$ is also regular.

Let Σ and Ω be two alphabets. Let $f : \Sigma \to \Omega$ be a function. Then f induces a homomorphism $\Sigma^* \to \Omega^*$. Let $w = a_1 a_2 \cdots a_n \in \Sigma^*$. Then

$$f(w) = f(a_1)f(a_2)\cdots f(a_n).$$

Let $L \subset \Sigma^*$. Then $f(L) = \{f(w) \in \Omega^* \mid w \in L\}$. If L is regular then $f(L)$ is also regular.

Let $L \subset \Omega^*$. Then $f^{-1}(L) = \{w \in \Sigma^* \mid f(w) \in L\}$. If L is regular then $f^{-1}(L)$ is also regular.

2 Context-free languages

Non-regular languages typically require us to maintain some count.

2.1 Pumping lemma

If L is a regular language then $\exists p \geq 0$, such that for all strings $w \in L$ with $|w| \geq p$, $\exists$ a partition of $w = xyz$ such that $|xy| \leq p$ and $|y| > 0$, and for all $i \geq 0$, $xy^i z \in L$.

Contrapositive form of the pumping lemma is quite useful to proving non-regularity of languages.

Closure properties can also be used in proving non-regularity of languages.

Some examples of non-regular languages are

1. $\{0^p \mid p \text{ is a prime}\}$.
2. $\{a^l b^m c^n \mid l + m \leq n\}$.

2.2 DFA minimization

The following algorithm produces a minimal DFA from a given DFA. Suppose the given DFA is $D = (Q, \Sigma, \delta, q_0, F)$.

1. Construct a table of all $\{p, q\}$ pairs where $p, q \in Q$.
2. Mark a pair $\{p, q\}$ if $p \in F$ and $q \notin F$ or vice versa.
3. Repeat the following until no more pair can be marked.
   (a) Mark $\{p, q\}$ if $\{\delta(p, a), \delta(q, a)\}$ is marked for some $a \in \Sigma$.
4. Two or more states are equivalent if they are not marked.

2.3 CFG

Context-free grammar recognizes a larger class of languages than regular languages. They find applications in programming languages and compiler design.

A CFG has terminal symbols (like alphabet) and variable symbols which are replaced by either variables or terminal symbols. One of the variables is the start variable S. Production rules dictate how variables get replaced.

For the non-regular language $\{0^n 1^{2n} \mid n \geq 0\}$, we have the following CFG:

Σ	Terminals	$\{0, 1\}$
V	Variables	$\{S, A, B\}$
P	Production rules	$S \to ASB \mid \epsilon$
		$A \to 0$
		$B \to 11$
S	Start variable	S

A CFG is represented by the tuple (V, Σ, P, S).

A language L is said to be context-free language (CFL) if $\exists$ a CFG G such that $L = L(G)$.

For the CFL $\{a^n b^m \mid n \geq 0, n \leq m \leq 2n\}$, we have the following CFG:

Σ	Terminals	$\{a, b\}$
V	Variables	$\{S, A, B\}$
P	Production rules	$S \to ASB \mid \epsilon$
		$A \to a$
		$B \to b \mid bb$
S	Start variable	S

For the CFL $\{w \in \{(,)\}^* \mid w \text{ is balanced}\}$, we have the production rule $S \to (S) \mid SS \mid \epsilon$.

Regular languages are context-free.

2.4 Parse tree

Let G be a CFG and $w \in L(G)$. A parse tree of w is a rooted ordered tree that represents the derivation of w with respect to G.

Some properties of parse trees are

1. Every internal node is a variable.
2. Every leaf node is either a terminal or ϵ.
   (a) If it is ϵ, it is the only child of its parent.
3. If an internal node is labelled R and A_1, A_2, ..., A_k are the children of R from left to right, then $R \to A_1 A_2 \cdots A_k$ is a production rule.
4. Concatenation of the leaves of a parse tree from left to right gives the string w.

There may exist multiple parse trees for a string w.r.t. a grammar.

The derivation of a string w w.r.t. a grammar G is a sequence of substitutions that yield w.

A leftmost derivation of w w.r.t. G is a derivation where in each step the left most variable of a string gets replaced.

A string $w \in L(G)$ is said to be ambiguous if it has 2 leftmost derivations. A grammar G is said to be ambiguous if $\exists$ some ambiguous string $\in L(G)$.

A CFL L is inherently ambiguous if all CFGs accepting L are ambiguous.

2.5 Chomsky normal form

A CFG $G = (V, \Sigma, P, S)$ is said to be in Chomsky normal form (CNF) if

1. every production rule is of the following types
   (a) $A \to BC$ where $B, C \in V$,
   (b) $A \to a$ where $a \in \Sigma$,
2. S does not appear on the right hand side of any production rule, and
3. the rule $S \to \epsilon$ may be present depending on whether $\epsilon \in L(G)$ or not.

Every CFL has a CNF grammar.

To reduce a CFG to CNF, use the following algorithm.

1. If S appears on the right hand side of any rule then add a new start variable S_0, and add the rule $S_0 \to S$.
2. Removing ϵ-transitions: remove $A \to \epsilon$ and for every occurrence of A on the right hand side of any rule, remove that occurrence and add a new rule.
3. Remove unit rules. $A \to B, B \to u$ to $B \to u, A \to u$.

4. Shortening the right hand side. Suppose $A \to u_1 u_2 \cdots u_k$. Add a variable $A_1 \to u_1 u_2$ so that $A \to A_1 u_3 \cdots u_k$.

5. A few more steps.

2.6 PDA

A pushdown automaton is like a ϵ-NFA with a stack. It has an input tape, a finite set of states and an unbounded stack.

Only the topmost element of the stack can be accessed at a given time. To access an intermediate element, all elements above it must be popped.

A step of computation by PDA is represented as $(p, a_i, X) \to (q, Y)$, which means the PDA moves from state p to q upon reading input a_i from the input tape and changes the topmost element of the stack from X to Y.

We may allow the stack to have a different alphabet Ω.

Stack operations:

1. $\epsilon \to Y$ means pushing Y onto the stack.
2. $X \to \epsilon$ means popping X from the stack.

A PDA may be represented by the tuple $(Q, \Sigma, \Omega, \delta, q_0, F)$. Due to non-determinism, there can be multiple transitions from the same tuple (p, a_i, X). A transition is also represented as

$$p \xrightarrow{a_i, X \to Y} q$$

Some special operations:

1. $p \xrightarrow{\epsilon, \epsilon \to \#} q$. Push # onto the stack without reading any input.
2. $p \xrightarrow{a, \epsilon \to A} q$. Push A onto the stack upon reading a.
3. $p \xrightarrow{\epsilon, \epsilon \to \epsilon} q$. Transition between states without reading any input and without changing the stack.

A language L is a CFL iff $\exists$ a PDA P such that $L = L(P)$. Basically, CFG is equivalent to PDA.

2.7 Closure

CFLs are closed under union, concatenation, Kleene star, reversal, homomorphisms and inverse homomorphisms.

Closure under concatenation is easily seen through additional rule $S \to S_1 S_2$. For Keene star, put additional rule $S \to S_1 S$. For reversal, we replace the rule $A \to A_1 A_2 \cdots A_k$ with the rule $A \to A_k \cdots A_2 A_1$.

If L_1 is a CFL and L_2 regular, then $L_1 \cap L_2$ is CFL.

CFLs are not closed under intersection, complementation and set difference.

As one application of closure, suppose we want to show that $L = \{a^n b^n a^{2m} b^{2m} \mid n, m \geq 0\}$ is a CFL. We use the fact that $L_1 = \{a^n b^n \mid n \geq 0\}$ is a CFL. Consider the homomorphism $h(a) = aa, h(b) = bb$. Then $h(L_1) = \{a^{2n} b^{2n} \mid n \geq 0\}$ so that $L = L_1 \cdot h(L_1)$ is a CFL.

2.8 DPDA

A deterministic pushdown automaton is a PDA such that $\delta(q, a_i, X)$ has at most one element and if $\delta(q, \epsilon, X) \neq \emptyset$ then $\delta(q, a, X) = \emptyset$ for all $a \in \Sigma$.

A language L is said to be a deterministic context-free language if $\exists$ a DPDA M such that $L = L(M)$.

Examples and non-examples:

1. $\{0^n 1^n \mid n \geq 0\}$ is a DCFL.
2. $\{w \in \{a, b\}^* \mid w = \text{rev}(w)\}$ is not a DCFL as it necessarily uses non-determinism to figure out the midpoint of the input string.
3. $L = \{a^n b^n c^n \mid n \geq 0\}$ is not a CFL. But $\bar{L}$ is a CFL and not a DCFL.

DCFLs are closed under complementation.

DCFLs are not closed under union or intersection.

Chomsky hierarchy: Regular $\subset$ DCFL $\subset$ CFL.

As an application, let L be the set of all strings in a, b, c such that the number of a's are the same as those of b's and c's. Consider $L_1 = \{a^* b^* c^*\}$ which is regular. See that $L \cap L_1 = \{a^n b^n c^n \mid n \geq 0\}$, which is not CFL. Therefore, L is not a CFL.

3 Turing machines

3.1 TM

Finite automata has finite control with no memory. Pushdown automata has finite control with memory in the form of stack.

A Turing machine has finite control and memory in the form of an unbounded tape which can be accessed using a tape head. It also has a designated accept state q_A and reject state q_R.

A typical transition of a Turing machine (TM) is

$$\delta(p, X) = (q, Y, L),$$

where X, Y are tape symbols and L denotes movement of tape head to left.

If Ω is the tape alphabet, we may write $\delta : Q \times \Omega \to Q \times \Omega \times \{L, R\}$.

Input is written on the tape itself and is followed by the blank symbol infinitely. Input alphabet $\Sigma \subset \Omega$. The blank symbol $\in \Omega - \Sigma$.

If the input head tries to move to the left of the leftmost cell of the tape, it stays at its current position.

A TM M is represented by a tuple $(Q, \Sigma, \Omega, \delta, q_0, q_A, q_R)$.

3.2 Configuration

A configuration of a TM M with respect to an input w is a snapshot of the machine consisting of

1. the current state,
2. the current contents of the tape,
3. the position of the tape head.

We represent a config by uqv where $q \in Q$, $u, v \in \Omega^*$ such that q is the current state, uv is the current contents of the tape and the tape head points to the first symbol of v.

Some standard configs are

1. start config: $q_0 w$, where w is the input,
2. accept config: $uq_A v$,
3. reject config: $uq_R v$.

Unlike other automata, a TM has a reject state. Of course, $q_A \neq q_R$.

M halts on w if it either accepts or rejects it. M is said to be a halting TM if for all $w \in \Omega^*$, M halts on w.

3.3 Recognizability and decidability

The language of a TM M is defined as $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

A language L is Turing recognizable (or simply recognizable) if $\exists$ a TM M such that $L = L(M)$.

A language L is Turing decidable (or simply decidable) if $\exists$ a halting TM M such that $L = L(M)$.

Consider $L = \{a^n b^n c^n \mid n \geq 0\}$. This is decidable because we can have a TM that works in the following way.

1. Checks whether the input is of the form $a^*b^*c^*$. If not, then reject.
2. Crosses off the first uncrossed a, the first uncrossed b, the first uncrossed c. If this fails, reject.
3. If all symbols are crossed off, then accept.

3.4 Variations of TM

Multiple tape TM are equivalent to single tape TM. The idea is to simulate a k-tape TM by using a single tape TM. We store all the contents of the k-tapes on the single tap. For every symbol $a \in \Omega$, add a symbol $\bar{a}$. Use this symbol to keep track of the tape head.

2-stack TM are equivalent to single tape TM. For example, suppose at some given time, the config is uqv. The two stacks can store u and v.

Queue machine is equivalent to single tape TM as well.

3.5 Non-determinism

In a non-deterministic TM, one can have multiple transitions from a given config. That is,

$$\delta : Q \times \Omega \to 2^{Q \times \Omega \times \{L,R\}}.$$

Consider $L = \{0^t \mid t \text{ is composite}\}$. This can be solved using a non-deterministic TM that basically does the following:

1. Write down n_1 number of 0's and n_2 number of 1's where $2 \leq n_1, n_2 \leq t$. This is non-deterministic.
2. Check if $n_1 \times n_2 = t$. If yes, then accept else reject.

Of course, the above problem can be solved deterministically as well.

3.6 Configuration graphs

A config graph $G_{M,x}$ is defined for a TM M w.r.t. an input x. Its vertices are the configurations of G w.r.t. x. There is an edge from config C_1 to C_2 if M can go $C_1 \implies C_2$ in one step.

Basically, a config graph is the graphical representation of computation of M on x.

A computation path is a path in $G_{M,x}$ from the start config. Clearly, M accepts x iff $\exists$ a computational path in $G_{M,x}$ from the start state to an accept state.

Some properties of config graphs are

1. If M is deterministic, the outdegree of every vertex in $G_{M,x} \leq 1$. If M is non-deterministic, the outdegree can be arbitrary.
2. Outdegree of accept and reject state configs are 0.
3. $G_{M,x}$ may be infinite. But if the size of the tape is bounded, $G_{M,x}$ is finite.
4. $G_{M,x}$ has a unique start and accept config.

The class of languages accepted by deterministic TM and non-deterministic TM are the same.

3.7 Closure

Decidable and recognizable languages are closed under union, concatenation, Kleene star and intersection.

Decidable languages (but not recognizable languages) are closed under taking complements.

A language L is decidable iff L and $\bar{L}$ are both recognizable.

3.8 Decidability properties

Let $A_{\text{DFA}} = \{\langle D, w\rangle \mid D \text{ is a DFA and } w \in L(D)\}$. Is A_{DFA} decidable? Yes, because we can implement the following algorithm in a halting TM.

1. Simulate D on w.
2. If D accepts w, then accept $\langle D, w\rangle$ else reject.

Similarly, A_{NFA} and A_{RE} are also decidable.

Let $E_{\text{DFA}} = \{\langle D \rangle \mid D \text{ is a DFA and } L(D) = \emptyset\}$. This is also decidable because we have the following algorithm:

1. Check if there is a path in the DFA from the start state to an accept state using BFS.
2. If no path exists then accept else reject.

$E_{\text{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ is also decidable.

The language $EQ_{\text{DFA}} = \{\langle D_1, D_2 \rangle \mid D_1 \text{ and } D_2 \text{ are DFA and } L(D_1) = L(D_2)\}$ is also decidable. However, EQ_{CFG} is not decidable.

Church-Turing thesis states that anything that can be computed can be computed by a Turing machine.

An example of non-Turing recognizable language is $L_d = \{s_j \mid s_j \notin L(M_j)\}$. This can be proven using Cantor's diagonal argument.

An example of un-decidable language is $A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ accepts } w\}$. But this is Turing recognizable. This also means $\overline{A_{\text{TM}}}$ is non Turing recognizable.

We define co-Turing recognizable by

$$\text{co-TR} = \{L \mid \bar{L} \text{ is TR}\}.$$

We saw that $A_{\text{TM}} \in$ TR and $\overline{A_{\text{TM}}} \in$ co-TR.

The language $H_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ halts on } w\}$ is un-decidable.

3.9 Reduction

Reduction in computer science means to convert one problem to another. For example, multiplication can be reduced to addition. Sorting can be reduced to finding minimum (or maximum).

Suppose $L_1 \leq_m L_2$.

1. If L_2 is decidable, then L_1 is decidable.
2. If L_1 is un-decidable, then L_2 is un-decidable.
3. If L_1 is not TR, then L_2 is not TR.

If $L_1 \leq_m L_2$, then L_2 is at least as hard as L_1.

Consider $E_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$. We can show that $\overline{A_{\text{TM}}} \leq_m E_{\text{TM}}$. Since $\overline{A_{\text{TM}}}$ is not TR, E_{TM} is not TR.

Some other applications would be

1. Show $E_{\text{TM}} \leq_m EQ_{\text{TM}}$ by producing a reduction function. Then EQ_{TM} will be not TR.

4 Complexity theory

Henceforth, we consider only decidable languages. We assume that all our TM are halting TM.

4.1 Space-time complexity

Let M be a deterministic TM.

1. The running time of M is said to be
$$t : \mathbb{N} \to \mathbb{N}$$
if for all $x \in \Sigma^*$, M halts in at most $O(t(|x|))$ steps.
2. The space required by M is said to be
$$s : \mathbb{N} \to \mathbb{N}$$
if for all $x \in \Sigma^*$, M uses at most $O(s(|x|))$ cells in its workspace.

We always count the amount of resource used as a function of the input length.

Since we use the O-notation we ignore the multiplicative constants and lower order terms.

Let $t : \mathbb{N} \to \mathbb{N}$, $s : \mathbb{N} \to \mathbb{N}$. Then we define

$$\text{TIME}(t(x)) = \{L \mid \exists \text{ a deterministic TM } M \text{ having running time } t(x) \text{ such that } L = L(M)\}$$
$$\text{SPACE}(s(n)) = \{L \mid \exists \text{ a deterministic TM } M \text{ using } s(n) \text{ space such that } L = L(M)\}$$

Let N be a non-deterministic TM.

1. The running time of N is said to be
$$t : \mathbb{N} \to \mathbb{N}$$
if for all $x \in \Sigma^*$, every computation path in N halts in at most $O(t(|x|))$ steps.
2. The space required by N is said to be
$$s : \mathbb{N} \to \mathbb{N}$$
if for all $x \in \Sigma^*$, every computation path of N uses at most $O(s(|x|))$ cells in its workspace.

Let $t : \mathbb{N} \to \mathbb{N}, s : \mathbb{N} \to \mathbb{N}$. Then we define

$$\text{NTIME}(t(x)) = \{L \mid \exists \text{ a NDTM } N \text{ having running time } t(x) \text{ such that } L = L(N)\}$$
$$\text{NSPACE}(s(n)) = \{L \mid \exists \text{ a NDTM } M \text{ using } s(n) \text{ space such that } L = L(N)\}$$

4.2 P and NP

We define

$$\text{P} = \bigcup_{k \geq 0} \text{TIME}(n^k)$$
$$\text{NP} = \bigcup_{k \geq 0} \text{NTIME}(n^k).$$

P is widely considered as class of problems having efficient solutions. NP is considered as class of problems whose solutions can be verified efficiently.

By "efficient" algorithms, we are talking about polynomial time algorithms, that is, $O(n^k)$.