CS 361: Theory of Computation

Finals Study Guide

Instructors: Shikha Singh

Languages

- An alphabet Σ is a finite set of symbols.
- Set of all finite strings over an alphabet Σ is denoted Σ^*.
- A language L is a subset of Σ^*.
- An empty string is a string containing no symbols and is denoted as ε.
- (**Operations on Languages**) Let L_1 and L_2 be two languages over the alphabet Σ.
  - *Union.* $L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}$
  - *Intersection.* $L_1 \cap L_2 = \{x \mid x \in L_1 \text{ and } x \in L_2\}$
  - *Complement.* $\overline{L_1} = \{x \in \Sigma^* \mid x \notin L_1\}$
  - *Concatenation.* $L_1 \circ L_2 = \{x \circ y \mid x \in L_1, y \in L_2\}$
  - *Kleene star.* ${L_1}^* = \{x_1 \circ x_2 \circ \cdots \circ x_k \mid k \geq 0, x_1, x_2, \ldots, x_k \in L_1\}$

Countability and Languages

- A function f is a bijection if it is both one-one and onto.
- An infinite set A is countable if there exists a bijection $f : A \to \mathbb{N}$.
- All finite sets are countable.
- The set Σ^*, is countable.
- The set of all languages over Σ (that is, the power set of Σ^*) is uncountable.

Regular Languages

- A Deterministic Finite Automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is the alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, q_0 is the start state, and F is the set of accept states. A DFA accepts a string $w = w_1 w_2 \ldots w_n$ if there exists a sequence of states starting with $r_0 = q_0$ and ending with $r_n \in F$ such that $\forall i$, $0 \leq i < n$, $\delta(r_i, w_i) = r_{i+1}$. The language of a DFA M, denoted $L(M)$ is exactly equal to the set of strings that M accepts.

- A language is regular if there is a deterministic finite automaton that recognizes it.
- **(Closure properties of regular languages.)** The class of regular languages are closed under union, concatenation, reverse, complement and Kleene star operations.
- A non-deterministic finite automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is the alphabet, $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the transition function, q_0 is the start state, and F is the set of accept states. A non-deterministic finite automaton accepts a string $w = w_1 \dots w_n$ if there exists a sequence of states $r_0, \dots r_n$ such that $r_0 = q_0$, $r_n \in F$ and $\forall i$, $0 \leq i < n$, $r_{i+1} \in \delta(r_i, w_i)$.
- For every NFA there is a DFA recognizing the same language.
- Regular expressions are built recursively starting from $\emptyset$, ε and symbols from Σ and closure under union $(R_1 \cup R_2)$, concatenation $(R_1 \circ R_2)$ and Kleene Star (R^*).
- A language is recognized by a DFA if and only if (iff) it is generated by some regular expression.
- All finite languages are regular.
- **(Pumping Lemma).** For every regular language L there is a pumping length p such that $\forall w \in L$, if $|w| \geq p$ then $w = xyz$ such that the following holds:
  - $|xy| \leq p$,
  - $|y| > 0$, and,
  - $\forall i \geq 0$, $xy^iz \in L$.
- **(Myhill-Nerode.)** Let L be a language over the alphabet Σ.
  - Two strings x and y are *indistinguishable with respect to* L, denoted $x \equiv_L y$, if for any $z \in \Sigma^*$, $xz \in L$ if and only if $yz \in L$.
  - The equivalence relation $\equiv_L$ partitions Σ^* into equivalence classes, where each equivalence class, denoted $[x]$, is the set of all strings that are indistinguishable, i.e., $[x] = \{w \in \Sigma^* \mid w \equiv_L x\}$.
  - If the relation $\equiv_L$ over Σ^* has k equivalence classes, then every DFA for L must have at least k states.
  - L is regular iff the relation $\equiv_L$ over Σ^* has a finite number of equivalence classes.
- Classic examples of non-regular languages are $\{a^nb^n \mid n \geq 0\}$ and $\{ww^R \mid w \in \{0,1\}^*\}$.
- Nonregularity of a language can be proved using either the pumping lemma or the Myhill Nerode theorem.

Context-free Languages

- A context-free grammar (CFG) is a 4-tuple (V, Σ, R, S), where V is a finite set of variables, with $S \in V$ the start variable, Σ is a finite set of terminals (disjoint from the set of variables), and R is a finite set of rules, with each rule consisting of a variable followed by $\rightarrow$ followed by a string of variables and terminals.
- Let $A \rightarrow w$ be a rule of the grammar, where w is a string of variables and terminals. Then A can be replaced in another rule by w, that is, uAv in a body of another rule can be replaced by uwv (we say uAv *yields* uwv, denoted $uAv \Rightarrow uwv$). If there is a sequence $u = u_1, u_2, \dots u_k = v$ such that for all i, $1 \leq i < k$, $u_i \Rightarrow u_{i+1}$ then we say that u derives v (denoted $u \overset{*}{\Rightarrow} v$.)
- If G is a context-free grammar, then the language of G is the set of all strings of terminals that can be generated from the start variable: $L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$.
- A parse tree of a string is a tree representation of a sequence of derivations; it is leftmost if at every step the first variable from the left was substituted.
- A grammar is called ambiguous if there is a string in a grammar with two different (leftmost) parse trees.
- A language is a context-free language (CFL) is a context-free grammar generates it.
- A pushdown automaton (PDA) is an NFA with a infinite stack. More formally, it is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q is the set of states, Σ is the input alphabet, Γ is the stack alphabet, q_0 is the start state, F is the set of accept states and the transition function $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$.
- A language is context-free if and only if some (non-deterministic) pushdown automaton recognizes it.
- Deterministic PDAs are not equivalent to non-deterministic PDAs.
- **(Closure properties of context-free languages.**)
  - Context-free languages are closed under union, Kleene star and concatenation.
  - Context-free languages are **not closed under** intersection and complement.
- **The intersection of a CFL and a regular language is context-free.**
  Even though we have not proved this in class, you can see why this is true by constructing a new PDA P', given the PDA P of the CFL, and a DFA M of the regular language. P' can simulate both P and M simultaneously and accept if both accept. Note that the stack of P' is the stack of P. The state of P' at any time is the pair (state of P, state of M). The transition function of P' follows both the transitions of P and M using its states and stack. The accept states of P' are those in which both the state of P and state of M are accepting.
- Classic non-context-free languages: $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ and $L = \{ww \mid w \in \{0,1\}^*\}$.

Turing Decidable and Recognizable Languages

- A Turing machine is a finite state machine with an infinite memory (tape). Formally, a Turing machine is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$. Here, Q is a finite set of states as before, with three special states q_0 (start state), q_{accept} and q_{reject}. The last two are called the halting states, and they cannot be equal. Σ a finite input alphabet. Γ is a tape alphabet which includes all symbols from Σ and a special symbol for blank $\sqcup$. Finally, the transition function is $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ where L, R mean move left or right one step on the tape.

- A Turing machine M accepts a string w (informally) if there is a sequence of configurations starting from q_0w and ending in a configuration containing q_{accept}, with every configuration in the sequence resulting from a previous one by a transition in δ of M. A Turing machine M recognizes a language L if M accepts x iff $x \in L$.

- Equivalent (not necessarily efficiently) variants of Turing machines: two-way vs. one-way infinite tape, multi-tape, and non-deterministic Turing machine.

- Any Turing machine can be encoded as a string over some alphabet Σ. Thus, the set of all Turing machines is infinitely countable.

- Church-Turing Thesis states that anything computable by an algorithm of any kind (our intuitive notion of algorithm) is computable by a Turing machine.

- A language L is Turing-recognizable (or recursively enumerable) if there is a Turing machine M such that M accepts x iff $x \in L$. M may reject or loop on any $x \notin L$.

- A language L is called decidable (or recursive) if there is a TM M such that M accepts x iff $x \in L$ and M rejects x if and only if $x \notin L$, i.e., M halts on all inputs.

- **(Closure properties of Decidable Languages.)** Decidable languages are closed under intersection, union, complementation, and Kleene star.

- If both L and $\overline{L}$ are Turing recognizable, then L is decidable.

- Decidable language examples: $\mathsf{A_{DFA}}$, $\mathsf{A_{NFA}}$, $\mathsf{A_{REX}}$, $\mathsf{E_{DFA}}$, $\mathsf{EQ_{DFA}}$, $\mathsf{ALL_{DFA}}$, $\mathsf{A_{CFG}}$, and $\mathsf{E_{CFG}}$.

- We proved $\mathsf{A_{TM}}$ is undecidability using proof by diagonalization. We used this to prove that $\overline{\mathsf{A_{TM}}}$ is not Turing recognizable.

- A function f is computable if there is a Turing machine that on input w halts with the description of $f(w)$ on its tape.

- There is a mapping reduction from A to B, written $A \leq_m B$ if exists a computable function $f : \Sigma^* \to \Sigma^*$, such that $x \in A \iff f(x) \in B$.

- To prove that B is undecidable, pick A which is undecidable and show that $A \leq_m B$.

- Undecidable language examples: $\mathsf{A_{TM}}$, $\mathsf{HALT_{TM}}$, $\mathsf{E_{TM}}$, $\mathsf{EQ_{TM}}$, $\mathsf{EQ_{CFG}}$, and $\mathsf{ALL_{CFG}}$.

Complexity Theory

- A Turing machine M runs in time $t(n)$ if for any input of length n the number of steps of M is at most $t(n)$.
- We say $f(n) = O(g(n))$ if there exists positive integers c and n_0 such that $f(n) \leq c(g(n)$ for every $n \geq n_0$. We say $f(n) = o(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- A language L is in the class P if there is a deternimistic Turing machine that decides L in polynomial time (that is, time $O(n^k)$ for some constant k).
- A language L is in the class NP if there is a non-deternimistic Turing machine that decides L in polynomial time (that is, time $O(n^k)$ for some constant k). Alternatively, L is in the class NP if there exists a polynomial-time verifier for it, that is, a polynomial-time TM V that given w and a certificate c, decides if $w \in L$ using c.
- Examples of languages in P: all regular and context-free languages, checking if a path exists in a graph, if a graph is connected, a number is composite, etc.
- Examples of languages in NP: all languages in P, Clique, Hamiltonian Path, SAT, etc.
- Major Open Problem: is $\mathsf{P} = \mathsf{NP}$?
- What we know: $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXPTIME}$ and $\mathsf{P} \subsetneq \mathsf{EXPTIME}$.
- A is polynomial-time reducible to B, written $A \leq_p B$ if there exists a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ such that $w \in A \iff f(w) \in B$.
- A language L is NP-hard if every language in NP reduces to L in polynomial time.
- A language is NP-complete it is both in NP and NP-hard.
- Cook-Levin Theorem states that SAT is NP-complete. The proof of this theorem can also be used to show that $\mathsf{3SAT}$ is NP-complete.
- Examples of NP-complete problems we discussed (along with the reduction used):
  - $\mathsf{3SAT} \leq_p \mathsf{CLIQUE}$
  - $\mathsf{CLIQUE} \leq_p \mathsf{VertexCover}$
  - $\mathsf{VertexCover} \leq_p \mathsf{IndSet}$
  - $\mathsf{3SAT} \leq_p \mathsf{HAMPATH}$ (directed)
  - $\mathsf{HAMPATH} \leq_p \mathsf{UHAMPATH}$ (proof in book)
  - $\mathsf{UHAMPATH} \leq_p \mathsf{UHAMCYCLE}$
  - $\mathsf{UHAMCYCLE} \leq_p \mathsf{TSP}$
  - $\mathsf{3SAT} \leq_p \mathsf{SUBSETSUM}$ (proof not discussed)