Midterm study sheet for CS3719

Regular languages and finite automata:

- An *alphabet* is a finite set of symbols. Set of all finite strings over an alphabet Σ is denoted Σ^*. A *language* is a subset of Σ^*. Empty string is called ϵ (epsilon).

- *Regular expressions* are built recursively starting from $\emptyset, \epsilon$ and symbols from Σ and closing under Union ($R_1 \cup R_2$), Concatenation ($R_1 \circ R_2$) and Kleene Star (R^* denoting 0 or more repetitions of R) operations. These three operations are called regular operations.

- A Deterministic Finite Automaton (DFA) D 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 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, denoted $\mathcal{L}(D)$ is the set of all and only strings that D accepts.

- Deterministic finite automata are used in string matching algorithms such as Knuth-Morris-Pratt algorithm.

- A language is called *regular* if it is recognized by some DFA.

- '**Theorem:** The class of regular languages is closed under union, concatenation and Kleene star operations.

- A *non-deterministic* finite automaton (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0 and F are as in the case of DFA, but the transition function δ is $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$. Here, $\mathcal{P}(Q)$ is the powerset (set of all subsets) of Q. A non-deterministic finite automaton *accepts* a string $w = w_1 \dots w_m$ if there exists a sequence of states $r_0, \dots r_m$ such that $r_0 = q_0$, $r_m \in F$ and $\forall i, 0 \leq i < m, r_{i+1} \in \delta(r_i, w_i)$.

- **Theorem:** For every NFA there is a DFA recognizing the same language. The construction sets states of the DFA to be the powerset of states of NFA, and makes a (single) transition from every set of states to a set of states accessible from it in one step on a letter following with all states reachable by (a path of) ϵ-transitions. The start state of the DFA is the set of all states reachable from q_0 by following possibly multiple ϵ-transitions.

- **Theorem:** A language is recognized by a DFA if and only if it is generated by some regular expression. In the proof, the construction of DFA from a regular expression follows the closure proofs and recursive definition of the regular expression. The construction of a regular expression from a DFA first converts DFA into a Generalized NFA with regular expressions on the transitions, a single distinct accept state and transitions (possibly $\emptyset$) between every two states. The proof proceeds inductively eliminating states until only the start and accept states are left.

- **Lemma** The *pumping lemma for regular languages* states that for every regular language A there is a pumping length p such that $\forall s \in A$, if $|s| > p$ then $s = xyz$ such that 1) $\forall i \geq 0, xy^iz \in A$. 2) $|y| > 0$ 3) $|xy| < p$. The proof proceeds by setting p to be the number of states of a DFA recognizing A, and showing how to eliminate or add the loops. This lemma is used to show that languages such as $\{0^n1^n\}, \{ww^r\}$ and so on are not regular.

Context-free languages and Pushdown automata.

- A *pushdown automaton* (PDA) is a "NFA with a stack"; more formally, a PDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q is the set of states, Σ the input alphabet, Γ the stack alphabet, q_0 the start state, F is the set of finite states and the transition function $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \to \mathcal{P}(Q \times (\Gamma \cup \{\epsilon\}))$.

- 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 $->$ followed by a string of variables and terminals.

- . Let $A \to w$ 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: 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 $u = u_1, u_2, \ldots u_k = v$ such that for all i, $1 \leq i < k$, $u_i \Rightarrow u_{i+1}$ then we say that u derives v (denoted $v \overset{*}{\Rightarrow} v$.) 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: $\mathcal{L}(G) = \{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}$. 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 called a *context-free language* (CFL) if there exists a CFG generating it.

- **Theorem** Every regular language is context-free.

- **Theorem** A language is context-free iff some pushdown automaton recognizes it. The proof of one direction constructs a PDA from the grammar (by having a middle state with "loops" on rules; loops consist of as many states as needed to place all symbols in the rule on the stack). The proof for another direction constructs a grammar that for every pair of states has a variable and a rule generating strings for a sequence of steps between these states keeping stack content.

- **Lemma** The *pumping lemma for context-free languages* states that for every CFL A there is a pumping length p such that $\forall s \in A$, if $|s| > p$ then $s = uvxyz$ such that 1) $\forall i \geq 0, uv^ixy^iz \in A$. 2) $|vy| > 0$ 3) $|vxy| < p$. The proof proceeds by analyzing repeated variables in large parse trees, setting the pumping length to $d^{|V|+1}$ where d is the length (number of symbols) of the longest rule. This lemma is used to show that languages such as $\{a^nb^nc^n\}, \{ww\}$ and so on are not regular.

- **Theorem** The class of CFLs is *not* closed under complementation and intersection (although it is closed under union, Kleene star and concatenation).

- **Theorem** There are context-free languages not recignized by any deterministic PDA. (No proof given in class).

Turing machines and decidability.

- A Turing machine is a finite automaton with an infinite memory (tape). Formally, a Turing machine is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$. 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.

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

- A Turing machine M *accepts* a string w if there is a sequence of configurations (state,non-blank memory,head position) starting from $q_0 w$ 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 it accepts all and only strings in L: that is, $\forall x \in \Sigma^*$, M accepts x iff $x \in L$. As before, we write $\mathcal{L}(M)$ for the language accepted by M.

- A language L is called *Turing-recognizable* (also *recursively enumerable, r.e*, or *semi-decidable*) if $\exists$ a Turing machine M such that $\mathcal{L}(M) = L$. A language L is called *decidable* (or *recursive*) if $\exists$ a Turing machine M such that $\mathcal{L}(M) = L$, and additionally, M halts on all inputs $x \in \Sigma^*$. That is, on every string M either enters the state q_{accept} or q_{reject} in some point in computation.