Review of Chapters 0-1

- See Midterm Review Slides
  - Emphasis on:
    - Sets, strings, and languages
    - Operations on strings/languages (concat, *, union, etc)
    - Lexicographic ordering of strings
    - DFAs and NFAs: definitions and how they work
    - Regular languages and properties
    - Regular expressions and GNFAs (see lecture slides)
    - Pumping lemma for regular languages and showing nonregularity

Context-Free Grammars (CFGs)

- CFG $G = (V, \Sigma, R, S)$
  - Variables, Terminals, Rules, Start variable
  - uAv yields uwv if $A \rightarrow w$ is a rule in G: Written as $uAv \Rightarrow uwv$
  - $u \Rightarrow^* v$ if u yields v in 0, 1, or more steps
  - $L(G) = \{w \mid S \Rightarrow^* w\}$
  - CFGs for regular languages: Convert DFA to a CFG (Create variables for states and rules to simulate transitions)
- Ambiguity: Grammar G is ambiguous if G has two or more parse trees for some string w in L(G)
  - See lecture notes/text/homework for examples
- Closure properties of Context-Free languages
  - Closed under $\cup$, concat, * *but not* $\cap$ *or complementation*.
  - See homework and lecture slides

Pushdown Automata (PDA)

- PDA P = $(Q, \Sigma, \Gamma, \delta, q_0, F)$
  - Q = set of states
  - Σ = input alphabet
  - Γ = stack alphabet
  - q_0 = start state
  - $F \subseteq Q$ = set of accept states
  - Transition function $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to Pow(Q \times \Gamma_\varepsilon)$
  - (current state, next input symbol, popped symbol) $\to$ {set of (next state, pushed symbol)}
  - Input/popped/pushed symbol can be ε
- Example PDAs for:
  - $\{w\#w^R | w \in \{0,1\}^*\}$, $\{ww^R | w \in \{0,1\}^*\}$, Palindromes

Context-Free Languages: Main Results

- CFGs and PDAs are equivalent in computational power
  - Generate/recognize the same class of languages (CFLs)
  1. If L = L(G) for some CFG G, then L = L(M) for some PDA M
     - Know how to convert a given CFG to a PDA
  2. If L = L(M) for some PDA M, then L = L(G) for some CFG G
     - Be familiar with the construction – no need to memorize the induction proof
- Pumping Lemma for CFLs
  - Know the exact statement: L CFL $\Rightarrow \exists p$ s.t. $\forall s$ in L s.t. $|s| \geq p$, $\exists$ u, v, x, y, and z s.t. $s = uvxyz$ and:
    1. $uv^ixy^iz \in L \ \forall\ i \geq 0$, **2.** $|vy| \geq 1$, and **3.** $|vxy| \leq p$.
- Using the PL to show languages are not CFLs
  - E.g. $\{0^n1^n0^n \mid n \geq 0\}$ and $\{0^n \mid n$ is a prime number$\}$

Turing Machines: Definition and Operation

- TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{ACC}, q_{REJ})$
  - Q = set of states
  - Σ = input alphabet not containing blank symbol "_"
  - Γ = tape alphabet containing blank "_", all symbols in Σ, plus possible temporary variables such as X, Y, etc.
  - q_0 = start state
  - q_{ACC} = accept and halt state
  - q_{REJ} = reject and halt state
  - Transition function $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- δ(current state, symbol under the head) = (next state, symbol to write over current symbol, direction of head movement)
  - *Configurations* of a TM, definition of language L(M) of a TM M

Decidable versus Recognizable Languages

- A language is Turing-recognizable if there is a Turing machine M such that L(M) = L
  - For all strings in L, M halts in state q_{ACC}
  - For strings not in L, M may either halt in q_{REJ} or loop forever
- A language is decidable if there is a "decider" Turing machine M that halts on all inputs such that L(M) = L
  - For all strings in L, M halts in state q_{ACC}
  - For all strings not in L, M halts in state q_{REJ}
- Showing a language is decidable by construction:
  - *Implementation level description of deciders*
  - E.g. $\{0^n1^n0^n \mid n \geq 0\}$, $\{0^n \mid n = m^2$ for some integer m$\}$, see text

Equivalence of TM Types & Church-Turing Thesis

- Varieties of TMs: Know the definition, operation, and idea behind proof of equivalence with standard TM
  - Multi-Tape TMs: TM with k tapes and k heads
  - Nondeterministic TMs (NTMs)
    - Decider if all branches halt on all inputs
  - Enumerator TM for L: Prints all strings in L (in any order, possibly with repetitions) and only the strings in L
- Can use any of these variants for showing a language is Turing-recognizable or decidable
- Church-Turing Thesis: Any formal definition of "algorithms" or "programs" is equivalent to Turing machines

Decidable Problems

- Any problem can be cast as a language membership problem
  - Does DFA D accept input w? Equivalent to:
    Is <D,w> in A_{DFA} = {<D,w> | D is a DFA that accepts input w}?
- Decidable problems concerning languages and machines:
  - A_{DFA}
  - A_{NFA} = {<N,w> | N is a NFA that accepts input w}
  - A_{REX} = {<R,w> | R is a reg. exp. that generates string w}
  - $A_{empty\text{-}DFA}$ = {<D> | D is a DFA and L(D) = ∅}
  - $A_{Equal\text{-}DFA}$ = {<C,D> | C and D are DFAs and L(C) = L(D)}
  - A_{CFG} = {<G,w> | G is a CFG that generates string w}
  - $A_{empty\text{-}CFG}$ = {<G> | G is a CFG and L(G) = ∅}

Undecidability, Reducibility, Unrecognizability

- A_{TM} = {<M,w> | M is a TM and M accepts w} is Turing-recognizable but not decidable (Proof by diagonalization)
- To show a problem A is undecidable, reduce A_{TM} to A
  - Show that if A was decidable, then you can use the decider for A as a *subroutine* to decide A_{TM}
  - E.g. Halting problem = "Does a program halt for an input or go into an infinite loop?"
  - Can show that the Halting problem is undecidable by reducing A_{TM} to A_H = { <M,w> | TM M halts on input w}
- A is decidable iff A and $\overline{A}$ are both Turing-recognizable
  - Corollary: $\overline{A}_{TM}$ and $\overline{A}_H$ are not Turing-recognizable