• (p. 170) TMs and Languages
  • \( L(M) = \) set of strings accepted by TM \( M \)
  • \( L \) is **Turing-recognizable** if some TM \( M \) accepts it
  • When \( M \) started, 3 outcomes: Accept, Reject, Loops
    • \( M \) can fail to accept if it enters \( q_{\text{reject}} \) or loops
  • (p. 170) \( M \) is a **decider** is it **never loops**
    • I.E. **always stops**, regardless of input string
    • I.e. always ends up in either \( q_{\text{accept}} \) or \( q_{\text{reject}} \)
  • (p. 170) \( L \) is **Turing-decidable** (or simply **decidable**) if some Turing Machine decides it.
• Examples
  • (p. 171 Ex. 3.7) \( A = \{ 0^k \mid k=2^n, n \geq 0 \} \)
    • Multiple iterations, each cuts \# 0s in half
  • (p.173 Ex. 3.9) \( B = \{ w#w \mid w \in \{0,1\}^* \} \)
  • (p. 174 Ex. 3.11) \( C = \{ a^i b^j c^k \mid ixj=k, i,j,k \geq 1 \} \)
  • (p.175 Ex. 3.12) \( E = \{ #x_1#x_2# \ldots #x_l \mid \text{no two } x\text{'s are equal} \} \)
• (p. 194) **Acceptance problem**: does some DFA accept some string?
  • Can we build a TM that:
    • given a representation for some FA and some string,
    • tell us if that FA accepts the string, or not
    • and do so in finite time
    • and never loop
• Define $A_{DFA} = \{<B,w>| B \text{ is a DFA that accepts } w\}$
  • $<B,w>$ is “encoding” of DFA $B$ and string $w$ in a way that a TM can “interpret” $B$’s processing of $w$
  • E.g. $<B>$ is a list of $B$’s 5 components
  • $A_{DFA}$ is set of all encoded DFAs & the strings they accept

• Is $A_{DFA}$ decidable?
  • Does there exist a TM that accepts all members of $A_{DFA}$ and rejects all other inputs?
    • I.e. does it always halt

• (p. 194) Theorem 4.1: $A_{DFA}$ is decidable
  • Proof: $M = \text{“On input } <B,w> \text{ where } B \text{ is a DFA } \& w \text{ a string”}$
    • $M$ receives a tape with $<B,w>$ on it
    • Determine if representation of $<B>$ is formatted ok
    • Simulate DFA $B$ on string $w$
      • Keep track of $B$’s current state and position into its input $w$ on $M$’s tape
      • Search for correct transition
      • Update state and index
    • If simulated $B$ ends in accept, accept. If it ends in nonaccept, reject.
      • Note: formatted $B$ always stops after finite # of steps
    • Thus so will TM
• Define $A_{NFA} = \{<B,w>| B \text{ is an NFA that accepts } w\}$

• (p. 195) Theorem 4.2: $A_{NFA}$ is decidable
  • Proof: $N = \text{“On input } <B,w> \text{ where } B \text{ is NFA & } w \text{ a string”}$
    • Convert NFA $B$ into equivalent DFA $C$
    • Encode $C$ and $w$ on tape as $<C,w>$
      • Having a multi-tape TM may be useful
    • Run machine $M$ from Theorem 4.1 on $<C,w>$
      • If $M$ accepts, $N$ accepts, else $N$ rejects
    • Note use of a “subroutine” $M$

• Define $A_{REX} = \{<R,w>| R \text{ is a regex that generates } w\}$

• (p. 196) Theorem 4.3 $A_{REX}$ is decidable
  • Proof: Convert $R$ into an NFA
    • Then run TM $N$
    • If $N$ accepts, then accept, else reject
• Define \( E_{DFA} = \{ <A> | A \text{ is a DFA where } L(A) = \emptyset \} \)
  • “E” for “empty”
  • I.e. the set of all DFAs that accept no strings

• (p. 196) Theorem 4.4 \( E_{DFA} \text{ is decidable} \)
  • Proof: Use the BFS algorithm starting on start state of \( A \)
    • Mark states that are reachable from start state
    • If any Final State is marked, reject
    • If not, accept
  • Again will halt since only finite # of states in any DFA

• Define \( EQ_{DFA} = \{ <A,B> | A,B \text{ both DFAs & } L(A) = L(B) \} \)
  • “EQ” stands for Equivalent
  • I.e. the set of all pairs of DFAs that are equivalent

• (p. 196) Theorem 4.5 \( EQ_{DFA} \text{ is decidable} \)
  • Proof:
    • Construct a new DFA \( C \) from \( A \) and \( B \) that
      • Accepts only those strings that are accepted by either \( A \) or \( B \), but not both
      • i.e. \( L(C) = (L(A) \cap \text{not}(L(B))) \cup \text{not}(L(A)) \cap L(B) \)
      • Called \textbf{Symmetric Difference}
      • If \( L(C) \) is empty then \( A \) & \( B \) gen same language
    • Then use machine from Theorem 4.4
• (p. 198) Decidable Problems re CFLs

• Define $A_{CFG} = \{<G, w> | G \text{ is a CFG that generates } w\}$

• (p. 198) Theorem 4.7 $A_{CFG}$ is a decidable language
  • If $G$ is in Chomsky Normal Form, any derivation of $w$ has $2n-1$ steps, where $|w| = n$
  • TM $S$
    • Convert $G$ to Chomsky
    • List all derivations with $2n-1$ steps
    • If any generate $w$, accept, else reject

• Define $E_{CFG} = \{<G> | G \text{ is a CFG & } L(G) = \emptyset\}$

• (p. 199) Theorem 4.8 $E_{CFG}$ is a decidable language
  • TM $R$
    • Mark all terminal symbols in $G$
    • Repeat until no new variables get marked
      • Mark any variable $A$ where $G$ has a rule $A \rightarrow U_1U_2...U_k$
        and each symbol $U_i$ has already been marked
    • If start variable not marked, accept, else reject
• Define $\text{EQ}_{\text{CFG}} = \{<G,H>| G \text{ & } H \text{ are CFGs, } & L(G)=L(H)\}$

• Cannot use DFA approach because CFLs not closed under complement or intersection & this is NOT decidable

• (p. 200) Theorem 4.9 **Every CFL is decidable**

  • Don’t want to try converting a PDA into an TM
  • Some branches of PDAs computation may go on forever, so TM can’t be a decider

  • Proof: Let $G$ be a CFG for $A$; TM $M_G$ is to decide $A$
    • Run TM $S$ on $<G,w>$
    • If it accepts, then accept, else reject

• Result: p.201 Fig. 4.10. Following are proper subsets of the next one
  • Regular languages
  • Context-Free languages
  • Decidable Languages
  • Turing-recognizable languages