Chap. 4,5 Review

- Algorithms created in proofs from prior chapters
  - (p. 55) Theorem 1.39: NFA to DFA
  - (p. 67) Lemma 1.55: Regex to NFA
  - (p. 69) Lemma 1.60: DFA to regex (through GNFA)
  - (p. 112) Lemma 2.21: CFG to PDA
  - (p. 121) Lemma 2.27: PDA to CFG
  - (p. 177) Theorem 3.13: Multi-tape TM to single tape
  - (p. 178) Theorem 3.16: NTM to TM

- Each of the proofs of decidable languages in Chap. 4 has an algorithm from the associated TM decider

- L is **Turing-recognizable** if some TM accepts any member, and never accepts a non-member
  - Halts on any member, but may not halt on non-members

- L is **Turing-decidable** if some TM accepts any member, and rejects all non-members
  - Halts for all inputs

- L is **co-Turing-recognizable** if some TM accepts any non-member, and never accepts any member
  - Halts on non-members, but may not halt on member

- L is **undecidable** if no TM decider exists
• (p. 209) L is decidable **iff** both Turing-recognizable and co-Turing-recognizable

• Interesting languages: languages whose members include descriptions (encodings) of machines
  • \(<M> = \text{“Encoding” of machine M as a string}\)
  • \(<M,w> = \text{“Encoding” of M and string w as a string}\)

• (p. 202) Diagonalization Method:
  • Compare 2 sets of possibly infinite size
  • If you can create table of 2 languages & can “correspond” every element of 1 set with element of other, then same size

• Decidable Regular Languages
  • (p. 194) \(A_{DFA} = \{<B,w>| B \text{ is a DFA that accepts w}\}\)
  • (p. 195) \(A_{NFA} = \{<B,w>| B \text{ is an NFA that accepts w}\}\)
  • (p. 196) \(A_{REX} = \{<R,w>| R \text{ is a regex that generates w}\}\)
  • (p. 196) \(E_{DFA} = \{<A>| A \text{ is a DFA where } L(A) = \emptyset\}\)
  • (p. 197) \(EQ_{DFA} = \{<A,B>| A,B \text{ both DFAs & } L(A) = L(B)\}\)
  • (Prob. 4.3) \(ALL_{DFA} = \{<A>| A \text{ a DFA and } L(A) = \Sigma^*\}\)
  • (Prob. 4.10) \(INFINITE_{DFA} = \{<A>| A \text{ a DFA, } L(A) \text{ is infinite}\}\)
  • (Prob. 4.11) \(INFINITE_{PDA} = \{<A>| A \text{ a PDA, } L(A) \text{ is infinite}\}\)

• Decidable Problems re CFLs
  • (p. 198) \(A_{CFG} = \{<G,w>| G \text{ is a CFG that generates } w\}\)
  • (p. 199) \(E_{CFG} = \{<G>| G \text{ is a CFG & } L(G) = \emptyset\}\)
• (p. 200) $\text{EQ}_{\text{CFG}} = \{<G, H>| G \& H \text{ are CFGs, }\& \text{ } L(G) = L(H)\}$
• (p. 200) Theorem 4.9 **Every CFL is decidable**
• Pr. 4.4 $\text{A}_\varepsilon_{\text{CFG}} = \{<G>| G \text{ is a CFG that generates } \varepsilon\}$

• Other Decidable problems
  • Pr. 4.5 $\text{ETM} = \{<M>| M \text{ a TM and } L(M) = \emptyset\}$
  • Pr. 4.11: $\text{INFINITE}_{\text{PDA}} = \{<M>| M \text{ a PDA and } L(M) \text{ is } \infty\}$
  • (p. 222,223) $\text{A}_{\text{LBA}} = \{<M,w>| M \text{ is LBA that accepts } w\}$
    • LBA is a TM that cannot move beyond initial input
    • Proof by showing # of configuration histories is finite

• **Undecidable Languages**: A decider does not exist.
  • (p. 202) $\text{HALT}_{\text{TM}} = \{<M,w>| M \text{ a TM that halts on } w\}$
  • (p. 207) $\text{A}_{\text{TM}} = \{<M,w>| M \text{ accepts } w\}$
  • (p. 217) $\text{ETM} = \{<M>| M \text{ a TM and } L(M) = \emptyset\}$
  • (p. 218) $\text{REGULAR}_{\text{TM}} = \{<M>| M \text{ a TM } \& L(M) \text{ is regular}\}$
  • (p. 219) $\text{L}_P = \{<M>| M \text{ a TM such that } L(M) \text{ has property } P\}$
  • (p. 220) $\text{EQ}_{\text{TM}} = \{<M_1,M_2>| M_1, M_2 \text{ TMs, } L(M_1) = L(M_2)\}$
  • (p. 222) $\text{A}_{\text{LBA}} = \{<M,w>| M \text{ an LBA that accepts } w\}$
  • (p. 223) $\text{E}_{\text{LBA}} = \{<M>| M \text{ an LBA where } L(M) \text{ is empty}\}$
  • (p. 225) $\text{ALL}_{\text{CFG}} = \{<G>| G \text{ is CFG where } L(G) = \Sigma^*\}$
  • (p. 228) $\text{PCP} = \{<P>| P \text{ instance of Post Correspondence Problem}\}$
• Result: p.201 Fig. 4.10. Following are proper subsets
  • RL subset of CFL subset of Decidable Languages subset of Turing-recognizable languages
• Undecidable Languages: No deciders exist
  • (p. 202) \( A_{TM} = \{<M,w>| M \text{ is a TM and } M \text{ accepts } w\} \)
  • Notional Proof:
    • Assume \( H \) a decider for \( A_{TM} \)
      • Accept if \( M \) accepts \( w \)
      • Reject if \( M \) does not accept \( w \)
    • Define \( D \) as machine with inputs \(<M>\)
      • Run \( H \) on \(<M,<M>>\)
      • Accept if \( H \) rejects \(<M>\), reject if \( H \) accepts \(<M>\)
    • Consider \( D(<D>) \)
      • Accepts if \( H \) rejects \(<D,<D>>, \text{ i.e. } D \text{ rejects } <D>\)
      • Reject if \( H \) accepts \(<D,<D>>, \text{ i.e. } D \text{ accepts } <D>\)
  • Tabular form of proof (p. 208)
    • Table rows = machines
    • Table columns = encodings of machines
    • One of the rows (and columns) is for \( D \)
  • Fig. 4.19: cell\([i,j]\) = running machine \( i \) on string \( j \)
  • Fig. 4.20: cell\([i,j]\) = running \( H \) on \(<<i>,j>\)
  • Fig. 4.21: look at row for \( D \) when it processes \(<D>\)
• (Chap. 5) **Reduction**: convert problem A into another problem B, where algorithm for B can solve A

![Diagram of reduction process]

If we chose A as a known undecidable language, then if R exists as decider for B, then we have built a decider for A - which we know is impossible

• **Typical Undecidability Proof for Language B:**
  • Assume Decider for B exists, and call it R
  • Choose some known undecidable language A
  • Design a reduction from any string \( w_A \) from A into a string \( w_B \) for B whereby the answer from R for \( w_B \) tells us the answer for \( w_A \)
  • Thus if decider R exists, so does one for Language A
Undecidable Problems about Turing Machines \( M \)

- (p. 216) \( \text{HALT}_{TM} = \{<M,w>|M\text{ halts on } w\} \)
  - Use \( A_{TM} \) for problem A

- (p. 217) \( E_{TM} = \{<M>|L(M)\text{ is empty}\} \)
  - Use \( A_{TM} \) for problem A

- (p. 219) \( \text{REGULAR}_{TM} = \{<M>|L(M)\text{ is regular}\} \)
  - Use \( A_{TM} \) for problem A

- (p. 220) \( \text{EQ}_{TM} = \{<M_1,M_2>|L(M_1) = L(M_2)\} \)
  - Use \( E_{TM} \) for problem A

- (p. 220) \( E_{LBA} = \{<M,w>|M\text{ is a LBA that accepts } w\} \)
  - Use \( A_{TM} \) for problem A

- (p. 225) \( \text{ALL}_{CFG} = \{<G>|G\text{ a CFG and } L(G) = \Sigma^*\} \)
  - Use \( A_{TM} \) for problem A

- (p. 227) Post Correspondence Problem

Pr. 5.1: \( \text{EQ}_{CFG} = \{<G_1,G_2>|L(G_1) = L(G_2)\} \)

Pr. 5.9: \( T = \{<M>|M\text{ accepts } w \text{ whenever it accepts } w^R\} \)
Problem Solving Steps:

- **Define the language precisely.**
  - Know what an element of the language is (as a string)
  - What properties does the string have to have
- For decision problems: **know what is to be decided**
  - You are looking for an algorithm/TM that *accepts* a string that is in the language, and rejects otherwise
  - You want to “write a program that always halts”
  - Typically, show how to
    - “Reduce” *any* string from language into a string for a language you know is decidable
    - Convert the answer from the known decider into an answer for the desired decider
- For undecidability problems, form a contradiction
  - Make sure you know what the language is (call it B)
  - Be explicit about what decider, if it exists, has to answer
  - Assume the decider exists (call it R)
  - Choose a undecidable language A , call “decider” for it as S, & build a reducer as above from any string in A to B
  - Show how answers from R then can answer S
  - Hint: sometimes reducer converts an input machine/grammar to a different machine/grammar