Chap. 4,5 Review

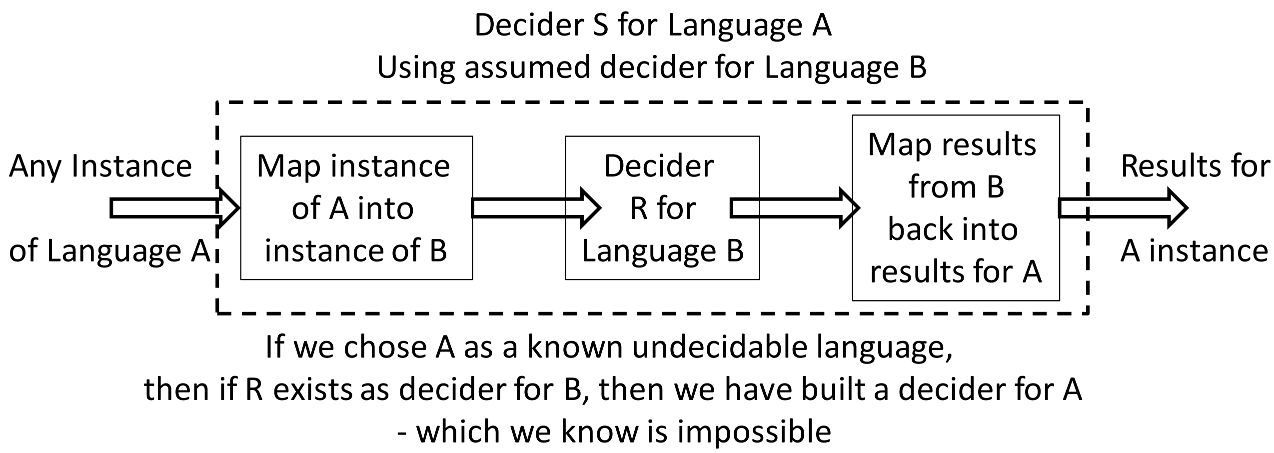
- Algorithms created in proofs from prior chapters
  - (p. 55) Theorem 1.39: NFA to DFA
  - (p. 67) Lemma 1.55: Regex to NFA
  - (p. 69) Lemma 1.60: DFA to regex (through GNFA)
  - (p. 112) Lemma 2.21: CFG to PDA
  - (p. 121) Lemma 2.27: PDA to CFG
  - (p. 177) Theorem 3.13: Multi-tape TM to single tape
  - (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> = "Encoding" of machine M as a string
  - <M,w> = "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 is a DFA that accepts w}
  - (p. 195) **A_{NFA}** = {<B,w>| B is an *NFA* that accepts w}
  - (p. 196) **A_{REX}** = {<R,w>| R is a regex that generates w}
  - (p. 196) **E_{DFA}** = {<A>| A is a DFA where L(A) = Φ}
  - (p. 197) **EQ_{DFA}** = {<A,B>| A,B both DFAs & L(A) = L(B)}
  - (Prob. 4.3) **ALL_{DFA}** = {<A>| A a DFA and L(A)=Σ*}
  - (Prob. 4.10) **$INFINITE_{DFA}$** = {<A>| A a DFA, L(A) is infinite}
  - (Prob. 4.11) **$INFINITE_{PDA}$** = {<A>| A a PDA, L(A) is infinite}
- Decidable Problems re CFLs
  - (p. 198) **A_{CFG}** = {<G,w>|G is a CFG that generates w}
  - (p. 199) **E_{CFG}** = {<G>|G is a CFG & L(G) = Φ}

- (p. 200) **EQ_{CFG}** = {<G,H>|G & H are CFGs, & L(G)=L(H)}
- (p. 200) Theorem 4.9 **Every CFL is decidable**
- Pr. 4.4 **$A\varepsilon_{CFG}$** = {<G>|G is a CFG that generates ε}
- Other Decidable problems
  - Pr. 4.5 **E_{TM}** = {<M>|M a TM and L(M)=Φ}
  - Pr. 4.11: **$INFINITE_{PDA}$** = {<M>| M a PDA and L(M) is ∞}
  - (p. 222,223) **A_{LBA}** = {<M,w>|M 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) **$HALT_{TM}$** = {<M,w>| M is a TM that halts on w}
  - (p. 207) **A_{TM}** = {<M,w>| M accepts w}
  - (p. 217) **E_{TM}** = {<M>| M is a TM and L(M)=Φ}
  - (p. 218) **$REGULAR_{TM}$**={<M>|M a TM & L(M) is regular}
  - (p. 219) **L_P** = {<M>| M a TM such that L(M) has property P}
  - (p. 220) **EQ_{TM}** = {<M1,M2>|M1, M2 TMs, L(M1)=L(M2)}
  - (p. 222) **A_{LBA}** = {<M,w>| M an LBA that accepts w}
  - (p. 223) **E_{LBA}** = {<M>| M an LBA where L(M) is empty}
  - (p. 225) **ALL_{CFG}** = {<G>| G is CFG where L(G)=Σ*}
  - (p. 228) **PCP** = {<P> | P 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 is a TM and M 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>>, i.e. D rejects <D>
        - Reject if H accepts <D,<D>>, i.e. D 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



- 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) **$HALT_{TM}$** = {<M,w>|M halts on w}
    - Use A_{TM} for problem A
  - (p. 217) **E_{TM}** = {<M>|L(M) is empty}
    - Use A_{TM} for problem A
  - (p. 219) **$REGULAR_{TM}$** = {<M>|L(M) is regular}
    - Use A_{TM} for problem A
  - (p. 220) **EQ_{TM}** = {<M1,M2>|L(M1) = L(M2)}
    - Use E_{TM} for problem A
  - (p. 220) **E_{LBA}** = {<M,w>|M is a LBA that accepts w}
    - Use A_{TM} for problem A
  - (p. 225) **ALL_{CFG}** = {<G>|G a CFG and L(G)=∑*}
    - Use A_{TM} for problem A
  - (p. 227) Post Correspondence Problem
  - Pr. 5.1: **EQ_{CFG}** = {<G1,G2>|L(G1) = L(G2)}
  - Pr. 5.9: T= {<M>|M accepts w 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