Decidability

Decidability

- We'll now take a look at Turing Machines at a high level and consider what types of problems can be solved algorithmically and what types can't:
  - What languages are Turing-decidable?
  - What languages are not Turing-decidable?
  - Is there a language that isn't even Turing-recognizable?

A Note on Notation

- We are going to follow Sipser's convention for describing Turing Machines at a high level
- If we want to describe a Turing Machine, M, that takes inputs A and B and solves a certain problem, we'll write this as:
  - M = "On input <A,B>, where A is a … and B is a …:
    - Enumerate the steps of the Turing Machine
    - If appropriate result occurs, accept. Otherwise, reject."

Using book's notation of putting the TM algorithm in quotes

Decidable Languages: DFA

- Acceptance problem for DFAs
  - $A_{DFA} = \{\langle B,w \rangle \mid B \text{ is a DFA that accepts input string } w\}$
  - This language is decidable.
    - A high level description of a TM, M, that decides A_{DFA}
      - M = “On input <B,w>, where B is a DFA and w is a string:
        - Simulate B on input w
        - If simulation ends in an accepting state, accept. If it ends in a nonaccepting state, reject.”

Using book’s notation of putting the TM algorithm in quotes

Decidable Languages: NFA

- Acceptance problem for NFAs
  - A_{NFA} = {<B,w> | B is an NFA that accepts input string w}
  - This language is decidable.
    - N = "On input <B,w>, where B is an NFA and w is a string:
      - Use subset construction to convert B to a DFA C
      - Use decider, M, for A_{DFA} on input <C, w>
      - If M accepts, accept
      - If M rejects, reject"

Decidable Languages: Regular Expressions

- Acceptance problem for Regular Expression
  - A_{REX} = {<R,w> | R is a regular expression that describes input string w}
  - This language is decidable.
    - P = "On input <R,w>, where R is a regular expression and w is a string:
      - Convert R to an NFA, A
      - Use decider, N, for A_{NFA} on input <A, w>
      - If N accepts, accept
      - If N rejects, reject"

Decidable Languages: Emptiness Test for DFAs

- Emptiness test for DFAs
  - $E_{DFA} = \{<A> \mid A \text{ is a DFA and } L(A) = \emptyset\}$
  - This language is decidable.
    - T = “On input <A>, where A is a DFA:
      - Mark start state of A
      - Repeat until no more states get marked
        - Mark any state that has a transition from a marked state
      - If no accept state is marked, accept…else reject.”

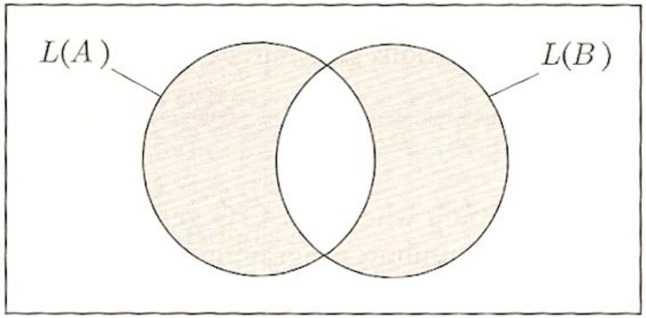
Decidable Languages: Equivalence of DFAs

- Equivalence test for DFAs
  - $EQ_{DFA} = \{<A,B> \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$
  - Important fact

Symmetric difference

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right)$$

L(C) = Ø iff L(A) = L(B)



Decidable Languages: Equivalence of DFAs

- Equivalence test for DFAs
  - $EQ_{DFA} = \{<A,B> \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$
  - This language is decidable.
    - F = “On input <A,B>, where A and B are DFAs:
      - Construct symmetric difference DFA, C, using Cartesian Product method
      - Run decider, T, for E_{DFA} on input <C>
      - If T accepts, accept.
      - If T rejects, reject.”

Decidable Languages: CFGs

- Acceptance test for CFGs
  - $A_{CFG} = \{<G,w> \mid G \text{ is a CFG that generates string } w\}$
  - Basic idea:
    - Try all derivations to see if G will generate w.
      - Requires infinite derivations, won't halt on non-accepted input string
    - However, if G is in Chomsky Normal Form, any derivation of w will take 2n–1 steps (for string of length n).
      - Only finite number of these.

Decidable Languages: CFGs

- Acceptance test for CFGs
  - $A_{CFG} = \{\langle G,w \rangle \mid G \text{ is a CFG that generates string } w\}$
  - This language is decidable.
    - S = “On input <G,w>, where G is a CFG and w is a string:
      - Convert G to Chomsky Normal Form
      - List all derivations with $2n-1$ steps, where $|w| = n$ (except when $n = 0$, in which case list all derivations with 1 step).
      - If any of these derivations generate w, accept, else reject.”

Decidable Languages: Emptiness Test for CFGs

- Emptiness test for CFGs
  - $E_{CFG} = \{<G> \mid G \text{ is a CFG and } L(G) = \emptyset\}$
  - Basic idea:
    - Cannot just test strings for membership in G using decider for A_{CFG}
      - Infinite number of w.
    - Instead…
      - Like E_{DFA} but from the opposite direction
      - Find variables that will generate a string of terminals.
      - If start variable is in this set, L(G) not empty.

Decidable Languages: Emptiness Test for CFGs

- Emptiness test for CFGs
  - E_{CFG} = {<G> | G is a CFG and L(G) = Ø}
  - This language is decidable.
    - R = "On input <G>, where G is a CFG:
      - Mark all terminals of G
      - Mark empty string symbol
      - Repeat until no more variables get marked
        - Mark any variable A where $A \rightarrow U_1U_2\ldots U_n$ and all U_i have been marked.
      - If start variable is not marked, accept…else reject."

Summary: Showing Decidability

- Avoid infinite loops
- Constructions allowed
  - Build DFAs
  - Minimize DFAs
  - Subset construction
  - Cartesian product construction
- Appeal to known results
  - Use already established decider TM as part of solution
    - Derive an "if and only if" relation

Decidable Languages: CFGs

- Every context-free language is decidable
  - Basic idea:
    - We could create a TM to simulate a PDA, but problem with infinite loop in using stack. Some strings that are not accepted might infinitely modify the stack, leading to non-halting behavior.
    - Instead, use the membership test for CFGs just developed.

Decidable Languages: CFGs

- Every context-free language is decidable
  - Let G be the CFG that generates L.
    - Must create a TM, M_G, that will accept strings in L, and reject strings not in L.
    - M_G = "On input string w:
      - Run decider, S, for A_{CFG} with input <G,w>
      - If S accepts, accept…else reject."

Language Bubble