8/10/11 CS 360 Lecture 17: Reductions

Last time: Decidability theory, Universal Turing machine

Goal in complexity theory: Classify problems/languages by computational hardness

- Decidably simple problems can be extremely hard to solve, or unsolvable!
- Especially those involving TMs since they can manipulate and simulate other TMs (there's no universal DFA or PDA).

Examples:

1. \( L_{DFA} = \{ <M, w> \mid M \text{ is a DFA} \} \) is decidable
   \[ L_{PDA} = \{ <M, w> \mid M \text{ is a PDA} \} \] is decidable

   Why? Simulate the DFA. The simulation will halt after \( 2^L \) steps.

   PDAs, though, can make \( \epsilon \)-moves (in addition to being non-deterministic), and might do so forever...

2. The Universal language
   \( L_u = \{ <M, w> \mid M \text{ is a Turing machine, } w \in L(M) \} \)
   is recursively enumerable but not decidable

   Why? RE, since the universal T.M. can simulate \( M \) on \( w \)

   Not decidable: If \( Au \) is an algorithm (halting T.M.) for \( L_u \), let
   \( Au : \) On input \( <M> \):
   Run \( Au(M, <M>) \)
   Accept if reject, reject if accepts.

   Running \( Au \) on its own encoding \( <Au> \) gives a contradiction!
Today: More examples of undecidable problems based on reductions

Example 1: Let $\text{HALT} = \{ <M, w> \mid M \text{ is a T.M., and } M \text{ halts on input } w \}$

Theorem: $\text{HALT}$ is recursively enumerable and not decidable.

Proof: For $\text{HALT} \neq \emptyset$, use the universal T.M. to simulate $M$.

Assume $\text{HALT}$ is decidable, with algorithm $R$.

Based on this algorithm, we construct an algorithm for $\text{L}_{\text{ALG}}$, which is a contradiction.

Algorithm S: On input $<M, w>$:
1. Run $R$ on $<M, w>$.
2. If $R$ rejects, then reject (M runs forever on $w$)
3. If $R$ accepts, then simulate $M$ on $w$ until it halts
4. If $M$ has accepted, accept. If $M$ has rejected, reject.

Corollary: $\text{L}_{\text{ALG}} = \{ <M, w> \mid M \text{ does not halt on input } w \}$ is not recursively enumerable.
Proof: Although $\text{L}_{\text{ALG}}$ is not the complement of $\text{HALT}$, it is close enough. Running the Turing machine for $\text{HALT}$ in parallel with a Turing machine for $\text{L}_{\text{ALG}}$ would give an algorithm for $\text{HALT}$, a contradiction.

Example 2: $\text{L}_{\text{ALG}} = \{ <M> \mid M \text{ is a T.M. that halts on every input} \}$ (i.e., an algorithm)

Theorem: $\text{L}_{\text{ALG}}$ is not recursively enumerable (its complement is).

Proof: Assume, for contradiction, we have a T.M. $R$ for $\text{L}_{\text{ALG}}$.

Turing machine $S$: On input $<M, w>$, with $M$ a T.M.:
1. Write down the encoding $<N>$ for the following T.M.:
   "$N$: On input $x$:
   1. Simulate $M$ on input $w$ for $|x|$ time steps.
   2. Halt if $M$ is not yet finished.
   Otherwise loop forever."
2. Run $R$ on this encoding; output $R(<N>)$.
Then $N$ halts on every input if and only if $M(w)$ runs forever.
Therefore $S$ is a Turing machine for $\text{HALT}$, a contradiction!
Example 3  Decision properties for machines

- **A** Fix \( w \in \Sigma^* \). Is \( w \in L(M) \)?
  - \( \text{M a DFA} \) is decidable
  - \( \text{M a PDA} \) is decidable
  - \( \text{M a T.M.} \) RE, not decidable
  - not R.E.
  - Rec. Enum.
  - not even R.E.

- **B** Is \( L(M) \) empty?
  - \( L(M) \) nonempty?
  - \( L(M) \) infinite?
  - \( L(M_1) = L(M_2) \) ?

- **C** Emptiness/nonemptiness:
  - **DFA**: To decide whether the language of a DFA is nonempty, run a graph traversal algorithm to see whether a final state is reachable from the initial state. Thus
    \[
    \text{Lempty, DFA} = \{ <M> | M \text{ is a DFA, } L(M) = \emptyset \}
    \]
    \[
    \text{Lnonempty, DFA} = \{ <M> | M \text{ is a DFA, } L(M) \neq \emptyset \}
    \]
  are both decidable languages.
  - **PDA**: For a PDA, convert it to a context-free grammar, and check whether the start symbol is generating.
  - **Turing machine**: Let
    \[
    \text{Lempty} = \{ <M> | M \text{ is a T.M., } L(M) \text{ is empty} \}
    \]
    \[
    \text{Lnonempty} = \{ <M> | M \text{ is a T.M., } L(M) \text{ is nonempty} \}
    \]
  - **Claim 1**: \( \text{Lnonempty} \) is recursively enumerable.
    - **Proof**: There is a nondeterministic TM, for \( \text{Lnonempty} \):
      On input \( <M> \) a T.M.
      1. Guess a string \( w \).
      2. Run \( M \) on \( w \) and accept if \( M \) accepts.
  - **Claim 2**: \( \text{Lempty} \) is not recursively enumerable.
    - **Proof**: Idea: Reduce from \( \text{Lnonempty} \).
      Say we have a procedure (Turing machine) for \( \text{Lempty} \).
      We use this to get a procedure for \( \{ <M, w> | w \notin L(M) \} \), contradicting
      On input \( <M, w> \), run our procedure on the following T.M. \( N \):
      \( N \) discards its input, and simply runs \( M \) on \( w \). Then \( L(N) \) is empty if and only if \( w \notin L(M) \).