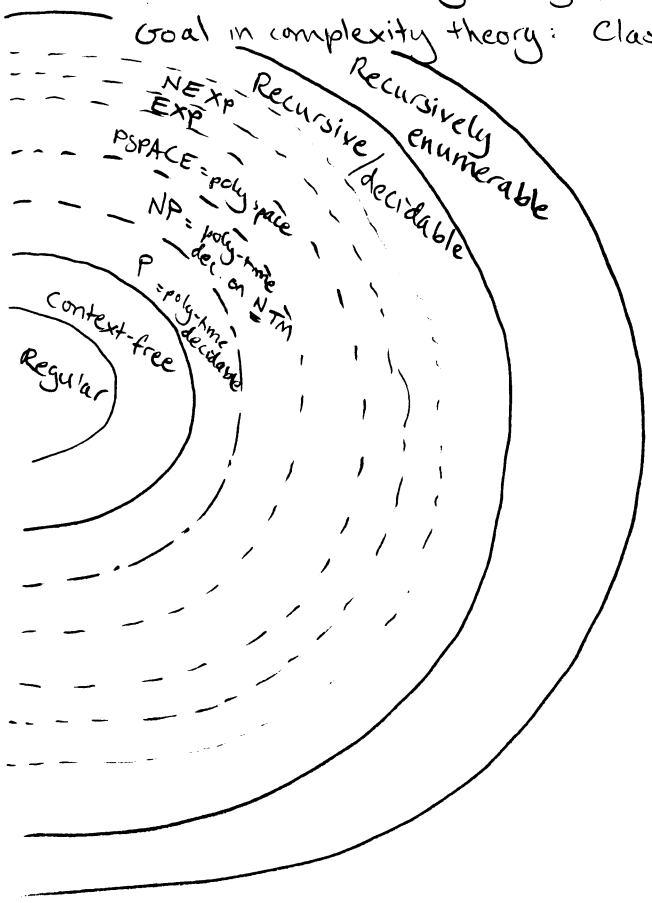
3/10/11 CS 360 Lecture 17 Reductions

Last time: Decidability theory, Universal Turing machine

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



Deceptably simple problems can be extremely hard to solve, or unsolvable!

—especially those involving TMs since they can manipulate & simulate other TMs (there is no universal DFA or PDA).

Examples:

① $L_{DFA} = \{ \langle M, w \rangle \mid M \text{ is a DFA and } w \in L(M) \}$ is decidable

$L_{PDA} = \{ \langle M, w \rangle \mid M \text{ is a PDA}, w \in L(M) \}$ is decidable

Why? Simulate the DFA. The simulation will halt after $|w|$ steps.

PDAs, though, can make ε-moves (in addition to being nondeterministic), and might do so forever...

② the universal language

$L_u = \{ \langle M, w \rangle \mid M \text{ is a Turing machine}, w \in L(M) \}$

is recursively enumerable but not decidable

Why? R.E. since the universal T.M. can simulate M on w

Not decidable: If A_u is an algorithm (halting T.M.) for L_u, let

$A_{\acute{u}}$: On input $\langle M \rangle$:

Run $A_u(M, \langle M \rangle)$

Accept if reject, reject if accepts.

Running $A_{\acute{u}}$ on its own encoding $\langle A_{\acute{u}} \rangle$ gives a contradiction!

Today: More examples of undecidable problems based on reductions

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

Theorem: HALT is recursively enumerable and not decidable.

Proof: For $HALT \in R.E.$, use the universal T.M. to simulate M. ✓

Assume HALT is decidable, with algorithm R.

Based on this algorithm, we construct an algorithm for L_U, which is a contradiction.

Algorithm S: On input $\langle M, w\rangle$:

1. Run R on $\langle M, w\rangle$.
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: $L_{\overline{HALT}} = \{\langle M, w\rangle \mid \begin{matrix} M \text{ is a T.M.} \\ M \text{ does not halt on input } w \end{matrix}\}$ is not recursively enumerable.

Proof: Although $L_{\overline{HALT}}$ is not the complement of HALT, it is close enough. Running the Turing machine for HALT in parallel with a Turing machine for $L_{\overline{HALT}}$ would give an algorithm for HALT, a contradiction. □

Example 2: $L_{ALG} = \{\langle M\rangle \mid M \text{ is a T.M. that halts on every input (ie., an algorithm)}\}$

Theorem: L_{ALG} is not recursively enumerable (its complement is).

Proof: Assume, for contradiction, we have a T.M. R for L_{ALG}.

Turing machine S: On input $\langle M, w\rangle$, with M a T.M.:

1. Write down the encoding $\langle N\rangle$ 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 $R(\langle N\rangle)$.

Then N halts on every input if and only if $M(w)$ runs forever.

Therefore S is a Turing machine for $L_{\overline{HALT}}$, a contradiction! □

Example 3 Decision properties for machines

		M a DFA	M a PDA	M a T.M.
(A)	Fix $w \in \Sigma^*$. Is $w \in L(M)$?	✓ decidable	✓ decidable	R.E., not decidable
(B)	Is $L(M)$ empty? Is $L(M)$ nonempty?	✓ decidable	✓ decidable	not R.E. Rec. Enum.
(C)	Is $L(M)$ infinite?	✓	✓	not even RE!
(D)	Is $L(M_1) = L(M_2)$?	✓	no (next class)	not RE. (today)

(B) 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

$L_{empty,DFA} = \{ \langle M \rangle \mid M \text{ is a DFA}, L(M) = \emptyset \}$

$L_{nonempty,DFA} = \{ \langle M \rangle \mid M \text{ is a DFA}, L(M) \neq \emptyset \}$

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

$L_{empty} = \{ \langle M \rangle \mid M \text{ is a T.M.}, L(M) \text{ is empty} \}$

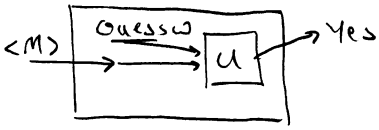
$L_{nonempty} = \{ \langle M \rangle \mid M \text{ is a T.M.}, L(M) \text{ is nonempty} \}$

Claim 1: $L_{nonempty}$ is recursively enumerable.

Proof: Here is a nondeterministic T.M. for $L_{nonempty}$:

On input $\langle M \rangle$ a T.M.:

1. Guess a string w.
2. Run M on w and accept if M accepts. □

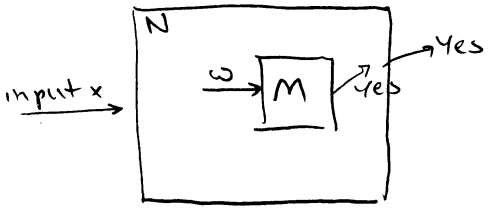


Claim 2: L_{empty} is not recursively enumerable.

Proof: Idea: Reduce from $\overline{L_u}$.

Say we have a procedure (Turing machine) for L_{empty}. We use this to get a procedure for $\{ \langle M, w \rangle \mid w \notin L(M) \}$, a contradiction.

On input $\langle M, w \rangle$, 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 $w \notin L(M)$ ✓ □