CSE 105
THEORY OF COMPUTATION

"Winter" 2018

http://cseweb.ucsd.edu/classes/wi18/cse105-ab/

Today's learning goals

Sipser Ch 5.1

- Define and explain core examples of computational problems, include A_{**}, E_{**}, EQ_{**}, $HALT_{TM}$ (for ** either DFA or TM)
- Explain what it means for one problem to reduce to another
- Use reductions to prove undecidability (or decidability)

Decidable	Undecidable
A_{DFA}	A_{TM}
E_{DFA}	A_{TM}^{C}
EQ_{DFA}	$HALT_{TM}$
	E_{TM}

Using reduction to prove undecidability

Claim: Problem X is undecidable

Proof strategy: Show that A_{TM} reduces to X.

Alternate Proof strategy: Show that $HALT_{TM}$ reduces to X.

Alternate Proof strategy: Show that E_{TM} reduces to X.

In each of these, have access to Genie which can answer questions about X.

$EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ TMs}, L(M_1)=L(M_2)\}$

A. Decidable
B. Undecidable
C. No way to tell

Give an example of a string in EQ_{TM} , and a string not in EQ_{TM}

Claim: ?? is no harder than EQ_{TM}

Given: Turing machine M, string w, magic genie for EQ_{TM}

Goal: ??

Building machines

In reduction proofs, we often need to build two different machines:

1. machine to decide problem
2. auxiliary machine to ask Genie about

E_{TM} reduces to EQ_{TM}

For machine that decides E_{TM}, what is input?

A. M
B. w
C. <M,w>
D. <M>
E. None of the above.

E_{TM} reduces to EQ_{TM}

$HALT_{TM}$ reduces to EQ_{TM}

- Input: <M,w>

- Goal: Accept if M halts on input w, Reject if M loops on input w

Auxiliary machine goal: build X based on M,w such that $L(X) = \Sigma^*$ if M halts on w, and $L(X) \neq \Sigma^*$ if M loops on w.

Recap

Decidable	Undecidable
A_{DFA}	A_{TM}
E_{DFA}	A_{TM}^{C}
EQ_{DFA}	$HALT_{TM}$
	E_{TM}
	EQ_{TM}

Which are recognizable?

Why care about Genies?

Reductions are central in

- (un)computability theory
- complexity theory
- cryptography

Central idea: how do we convert information about one problem to information about another?

Complexity theory

Chapter 7

In the "real world", computers (and Turing machines) don't have infinite tape, and we can't afford to wait unboundedly long for an answer.

"Decidable" isn't good enough – we want "Efficiently decidable"

Not just decidable …

- For a given **algorithm** working on a given **input**, how long do we need to wait for an answer? How does the running time depend on the input in the worst-case? average-case? Expect to have to spend more time on larger inputs.

Measuring time

- For a given **algorithm** working on a given **input**, how long do we need to wait for an answer? Count steps! How does the running time depend on the input in the worst-case? average-case? Big-O

Can we detect problems that are **efficiently solvable**?

Time complexity

For M a deterministic decider,

its **running time** or **time complexity** is the function

f: N → R^+ given by

f(n) = maximum number of steps M takes before halting,
over all inputs of length n.

worst-case analysis

Plus, instead of calculating precisely, **estimate** f(n) by using big-O notation.

Time complexity classes

TIME(t(n)) = { L | L is decidable by a TM running in O(t(n)) }

- **Exponential** $EXPTIME = \bigcup_k TIME(2^{n^k})$

Brute-force search

- **Polynomial** $P = \bigcup_k TIME(n^k)$

Invariant under many models of TMs

- **Logarithmic**

May not need to read all of input

$$P = \bigcup_k TIME(n^k)$$

Why is it okay to group all polynomial running times?

- Contains all the "feasibly solvable" problems.
- Invariant for all the "usual" deterministic TM models
  - multitape machines (Theorem 7.8)
  - multi-write

Working with P

- Problems encoded by languages of strings
  - Need to make sure coding/decoding of objects can be done in polynomial time.

- Algorithms can be described in high-level or implementation level

CAUTION: not allowed to guess / make non-deterministic moves.

Next time

Pre-class reading skim Chapter 7