COMS W3261, Lecture 11:

Reductions & Time Complexity

Announcements: HW #6, due Monday 8/9 @ 11:59 PM. ← do this first

Final Exam: 8/10, 8/11.

Posted Exam topics on Ed.

Review Sessions — 1-4 PM on Monday 8/9 (in-person)
5-8 PM —— " —— (virtual)

Readings: Sipser 5.1 (Undecidability & Reductions)
Sipser 7.1-7.3 (Time Complexity, P and NP)

Today:

1. Review
2. Reductions & Undecidable Languages
3. Big-O and Time Complexity
4. P and NP (P = NP?)

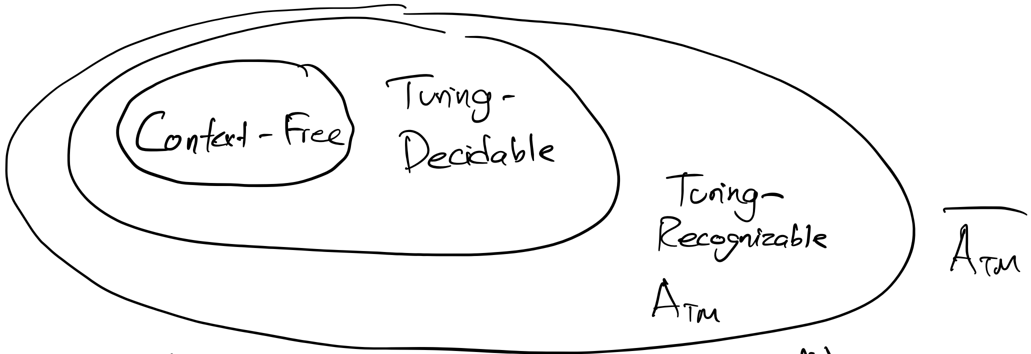
---

1. Review

Decidable languages:

- $A_{DFA} = \{\langle A, w\rangle \mid \langle A\rangle \text{ encodes a DFA, } A \text{ accepts } w\}$
- A_{NFA}, A_{REX}, A_{CFG} decidable
  - A_{NFA}, A_{REX} → reducing to DFA
  - A_{CFG} → stated as fact.
- $E_{DFA} = \{\langle A\rangle \mid A \text{ is a DFA, } L(A) = \emptyset\}$
- $EQ_{DFA} = \{\langle A, B\rangle \mid A, B \text{ are DFA, } L(A) = L(B)\}$

E_{CFG}, EQ_{CFG} decidable.



- A set is countable if it admits a 1-to-1 mapping to $\mathbb{N} = 1, 2, 3, \ldots$
- The set of TMs was countable.
- The set of infinite binary strings was uncountable
  ↪ the set of languages over any nonempty alphabet was uncountable

$\therefore$ No 1-to-1 mappin between TMs and languages.
$\therefore$ We can't map TMs onto the set of languages they recognize.

- $A_{TM} = \{\langle A, w\rangle \mid A$ is a TM that accepts $w\}$ is recognizable, but not decidable (we showed if A_{TM} were decidable, we could build a paradoxical TM.)
- $\overline{A_{TM}}$ is unrecognizable. (If we could recognize both A_{TM}, $\overline{A_{TM}}$, then we could decide A_{TM}.)

Now: build a family of undecidable languages.

---

2. Reductions & More Undecidable Languages.

Idea: Laziness. Build solutions to hard problems using known solutions for easy problems.

Prove: "If I can do B, then I can do A."

Know: "I can do B."

$\therefore$ I can do A!

"A is a hard problem" (A is undecidable, unrecognizable)

Prove: "If I could solve B, I could solve A"

$\therefore$ I can't solve B
(B is a hard problem.)

The Halting Problem.

Theorem. $HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on } w\}$.

Proof. By contradiction, reducing A_{TM} to $HALT_{TM}$. We show that if $HALT_{TM}$ were decidable, A_{TM} would be decidable.

Assume that some TM R decides $HALT_{TM}$. Then we can build a new machine that decides A_{TM}:

M_1 = "On input $\langle M, w\rangle$, where M is a TM, w a string:

1. Simulate R on $\langle M, w\rangle$. Reject if R rejects.
2. If R accepts, simulate M on w until it halts, then accept/reject if M accepts/rejects."

(Why does this work?

$M(w)$ runs forever — reject (step 1)
$M(w)$ stops, accepts — accept (step 2)
$\hookrightarrow$ rejects — reject (step 2).

M_1 Always stops — because R always stops.)

A_{TM} is not decidable, so the existence of R is a contradiction. $\blacksquare$

Idea: Show "$HALT_{TM}$ decidable $\rightarrow$ A_{TM} decidable."
We know A_{TM} not decidable. So this $\rightarrow$
$HALT_{TM}$ not decidable.

M_1 is our hypothetical decider for A_{TM}.

Q. Is $HALT_{TM}$ recognizable? Yes — simulation works here.

$\boxed{Q.}$ Is $\overline{HALT_{TM}}$, the language of programs that run forever on the given input, recognizable?

No — If $HALT_{TM}$, $\overline{HALT_{TM}}$ recognizable $\rightarrow$ $HALT_{TM}$ decidable. (contradiction.)

Moral: It is impossible to write an infinite loop detector.

Example: $E_{TM} = \{\langle M\rangle \mid M \text{ is a TM}, L(M) = \emptyset\}$ is undecidable.

Proof: We'll show that if E_{TM} were decidable, we could decide A_{TM} — a contradiction. Suppose S decides E_{TM}. We'll build a decider T for A_{TM}.

$T =$ "On input $\langle M, w\rangle$, where M is a TM, w a string,

- Use $\langle M\rangle$ to build a new TM, M', that rejects all strings $x \neq w$, and on w will simulate $M(w)$ and accept/reject if $M(w)$ accepts/rejects.
- Now: Simulate S on M'.
  If S accepts $\langle M'\rangle$, then $L(M') = \emptyset$, and thus $M(w)$ does not accept. Reject.
  If S rejects $\langle M'\rangle$, then $L(M') = \{w\}$, and thus

$M(w)$ accepts. Accept."

If S decides E_{TM}, then T decides A_{TM}, a contradiction.

(1) If M accepts $w \longrightarrow T(\langle M, w \rangle)$ accepts.

(2) If M $\neg$accepts $w \longrightarrow T(\langle M, w \rangle)$ rejects.

Why no infinite loop? We build M; but we never run M!

Example. $EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2$ are TMs and $L(M_1) = L(M_2) \}$

Proof. We show that if EQ_{TM} is decidable, then E_{TM} is decidable — a contradiction. Suppose some TM R decides EQ_{TM}. Then the following TM decides E_{TM}:

$S =$ " On input $\langle M \rangle$, where M is a TM

1. Run R on input $\langle M, M_1 \rangle$, where M_1 is a TM that always rejects. Accept/reject if R accepts/rejects."

$\square$

If decide $EQ_{TM} \to$ decide $E_{TM} \to$ decide A_{TM}

paradox machine. X

Rice's Theorem. Let P be a language of TM descriptions such that

(1) P contains some, but not all, TMs. (P nontrivial)

(2) P captures some property of the language recognized by its input: If $L(M_1) = L(M_2)$, $\langle M_1 \rangle \in P \leftrightarrow \langle M_2 \rangle \in P$.

Then P is undecidable.

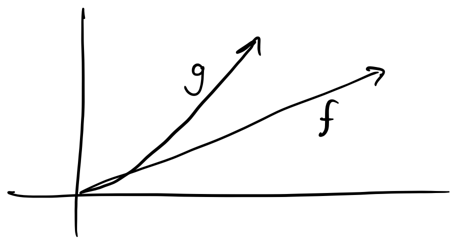
("All nontrivial properties of TMs are undecidable!")

Break — 10 m — back at 11:24 AM.

3. Big-O notation — measuring complexity.

Recall: Asymptotic analysis —

"roughly comparing functions."



Def. Let f and g be functions $f, g : \mathbb{N} \to \mathbb{R}^+$. We say

$f(n) = O(g(n))$ if there exist positive integers n_0 and c such that $f(n) \le c \cdot g(n)$ for all $n \ge n_0$.

"In the long run, f is at most some constant times g."

" f is not much bigger than, ~smaller~ than g."

$f(n) = \Omega(g(n))$ if there exist positive numbers n_0 and c such that $f(n) \ge c \cdot g(n)$ for all $n \ge n_0$.

"In the long run, f is at least some constant times g."

Examples.

$$\frac{n}{2} = O(n).$$

$$5n = O(n).$$

$$10 = O(\log_2(n)) = O(1).$$

$$n = \Omega(\log_2(n)) = \Omega(1).$$

$$16n^2 + n + 4 = O(n^2).$$

Def. Let M be a deterministic TM that halts on all inputs.
The <u>running time</u> or <u>time complexity</u> of M is a function $f: \mathbb{N} \to \mathbb{N}$, where $f(n)$ indicates the maximum number of steps M uses on any input of length n.

set of languages

Def. We define the complexity class $TIME(t(n))$ to be the set of all languages that can be decided by an $O(t(n))$ Turing Machine.

<u>Example:</u> Algorithms for $A = \{0^k 1^k \mid k \geq 0\}$.

(Input length is n.)

<u>Approach 1:</u>

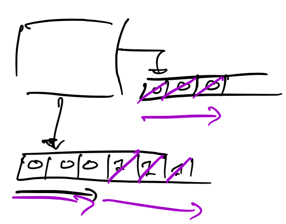
M_1: "On input w,

Time $O(n)$. 1. Scan and reject if any 0 appears to the right of any 1.

each shuttle: time $O(n)$. # of shuttles: at most $O(n)$ $= O(n^2)$.

2. Shuttle back + forth, crossing off pairs of 0's and 1's. Accept if the number of 0's equals the number of 1's, reject otherwise."

Total: $O(n^2) + O(n) = O(n^2)$.



<u>Approach 2: Better — with a 2-tape TM.</u>

M_2 = "On input w:

Time $O(n)$ 1. Scan across the tape + reject if we find bad input.

Time $O(n)$ 2. Scan until we see a 1; copy all 0's to our second tape.

Time $O(n)$ 3. Scan all the 1's, crossing off a 1 on the input tape for each 0 on the second tape. Accept if and only if the number of 1's and the number of 0's is equal."

Total time: $O(n)$, linear.

Takeaway: Multitape TMs may not be able to decide more languages, but they might be faster than single-tape TMs.

Exercises: Can you beat $O(n^2)$ on a single-tape TM?
Can you prove $O(n)$ is impossible with one tape?

P: Polynomial Time.

Def. P is the class of all languages decidable in polynomial time,
In other words, $P = \bigcup_{k \geq 0} TIME(n^k)$.

Why this class? $TIME(n^c) \subseteq TIME(n^{c+1})$

Idea: polynomials grow much more slowly than exponentials.

At $n = 1000$, $n^3 = 1$ billion
$2^n \geq$ number of atoms in the universe.

Idea: polynomial · polynomial = polynomial.
↳ Problems in P can use each other as subroutines.

"I can solve A by solving B n^c times."
"B takes time n^d."

↳ I can solve A in time $O(n^c \cdot n^d) = O(n^{c+d})$.
so $A \in P$.

Idea: Polynomials tend to have small exponents "in practice."

Brute force solutions — often exponential $(\Omega(2^n))$
"Smart" solutions — often small polynomials.

Roughly: "P is the class of efficiently decidable languages."

Some example problems in P:

- All context-free languages. (Sipser 7.2)
- PATH = $\{\langle G, s, t\rangle \mid$ There is a path from s to t in G.$\}$
- RELPRIME = $\{\langle x, y\rangle \mid x, y$ are relatively prime$\}$
- many, many, many more.
- MULT = $\{a^i b^j c^k \mid ij = k\}$.

Sipser: "All reasonable deterministic computational models are polynomially equivalent."

convert programs back and forth / simulate each other with polynomial increase in runtime.

This means P is "model-independent."

NP: Nondeterministic Polynomial Time

Idea: A problem is verifiable if you can show me some certificate / evidence that a given string is in the language. (This doesn't mean it's easy to decide.)

Example –

SUDOKU = $\{\langle P\rangle \mid P$ is a sudoku and P is solvable.$\}$

Easy to verify, hard to solve.

certificate for $\langle P \rangle$: a solved puzzle.

<u>Def.</u> A verifier for a language A is an algorithm (TM) V, where

$$A = \{ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some certificate } c \}.$$

We'll say V is a polynomial time verifier if it runs in time polynomial in the length of the input w.

($O(|w|^c)$ for some c.)

<u>Def.</u> NP is the class of all languages that have polynomial-time verifiers.

Essentially – "all languages where membership $w \in L$ can be efficiently proved." (with certificate).

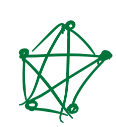
<u>Examples:</u>

Sudoku — certificate is a solved puzzle.

Subset Sum $= \{ \langle S, t \rangle \mid S$ is some set of numbers, some subset of S adds to the target $t. \}$

(certificate: a subset that sums to t.)

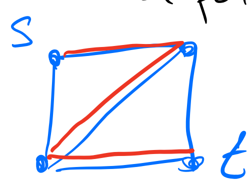
k-Clique $= \{ \langle G \rangle \mid G$ has a <u>complete subgraph</u> of size $k. \}$



(certificate: a k-clique)

HAMPATH $= \{ \langle G, s, t \rangle \mid G$ is a directed graph with a Hamiltonian Path from s to t —

a path that touches every node exactly once?



(certificate: a working path.)

P ⊆ NP. Why? Imagine some language in P.
There exists some TM that decides the language in polynomial time. Now — an accepting computation for a string in the language is itself a certificate.

NP ⊆ P? Seems very unlikely. Could it really be true that every problem where the answer can be efficiently proved correct is also easy to solve? Probably not.

Conjectured P ≠ NP.

We don't know. P = NP?

(NP = all languages decided by a Nondeterministic TM in polynomial time.)

What have we learned?

- Formal science for computers
- Languages = sets of strings ≈ concepts.
- Automata = math machines.
- Computation has limits.

- More techniques for solving formal problems fast.
  CSOR W4231 — Algorithms.
- P, NP, and beyond. Computability and the universe of problems.

COMS W4236 — Complexity.

— COMS W4252 — Computational Learning Theory. How can we train computers to categorize things?

Thank you!