Time complexity classes, $\mathtt{P}$ and $\mathtt{NP}$

Textbook: Chapter 7.2, 7.3

The class $\mathtt{P}$

- Recall: $\mathtt{TIME}(f(n))$ is the set of all *languages* decidable in $O(f(n))$ deterministic time
- We find it interesting to study the differences between polynomial and nonpolynomial time classes: Therefore we create the set of all languages decidable in polynomial deterministic time

Def: P. The set of all language decidable in deterministic polynomial time is called P, and is defined as below.

$$P = \bigcup_k \mathtt{TIME}(n^k)$$

Verifiers

- What if a TM got some arbitrary extra "proof" or "certificate" that helped it out? Equivalently, given an answer to a question, can we prove that it is true faster than we could find the solution?
- For instance, if we are trying to prove a number isn't prime, a certificate could be a list of its factors: The "verification" algorithm would take time proportional to the number of factors (very fast)!

Def: A **verifier** for a language A is an algorithm V where

$$A = \{w : V \texttt{ accepts } \langle w, c \rangle \texttt{ for some "certificate" } c\}$$

Measuring Verifiers

We measure the time of a verifier only in terms of $|w|$ (**not** $|c|$: This is given "magically"), so a **polynomial time verifier** runs in polynomial time with respect to $|w|$.

- A verifier takes in some candidate solution and accepts iff it is indeed a true solution

Note: An nTM can nondeterministically "guess" the certificate then simulate a verifier TM given it. Therefore, nTMs can simulate n deterministic verifier steps in $O(n)$ nondeterministic time.

The class NP

- Just like we used TIME to define P, we can use NTIME to define the set of all languages decidable in **nondeterministic polynomial time**

Def: NP. The set of all languages decidable by a nondeterministic TM in polynomial time is called NP and is defined as follows.

$$NP = \bigcup_k \mathtt{NTIME}(n^k)$$

- **Equivalently**, NP is the set of all problems that can be verified in deterministic polynomial time

Does $P = NP$?

- Does $P = NP$?
- Can all problems whose solutions can be checked in polynomial time be solved in polynomial time?

Nobody knows! But we think $P \neq NP$

- No-one has proven it or disproven it yet
- Alternatively, it could be a statement that is true but has no proof

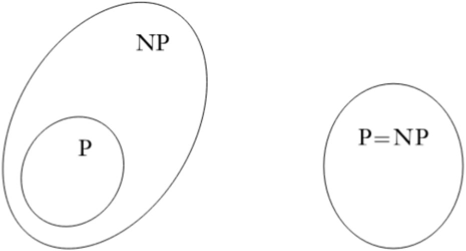

FIGURE 7.26
One of these two possibilities is correct

Properties of P and NP

- The best known algorithm for simulating an nTM on a dTM is $O(2^n)$
- So NP can be no harder than exponential time

$$NP \subseteq \texttt{EXPTIME} = \bigcup_k \texttt{TIME}(2^{n^k})$$

NP **may be a smaller deterministic time complexity class!**

- All reasonable models of computation are polynomial-time simulatable by each other, so P and NP are consistent across them
- This seems to be a fundamental complexity barrier between deterministic and nondeterministic computation

Polynomial-time mapping reducibility

- We want to be able to do reductions on time complexity classes
- However, we can't just use any TM! What if it took exponential time?

Def: A function $f : \Sigma^* \to \Sigma^*$ is **polynomial-time computable** if some polynomial-time TM exists that takes in some w and halts with just $f(w)$ on its tape.

- Now we can do language reductions while staying in P!

Def: Language A is **polynomial-time mapping reducible** to language B, written $A \leq_P B$, if a polynomial-time computable function f exists such that, for every w,

$$w \in A \iff f(w) \in B$$

NP-completeness

- Recall: A machine is Turing-complete if it can solve any problem solvable by a Turing machine
- Similarly, a problem is NP-complete if a polynomial-time solution to it would solve any problem $\in NP$ in polynomial time
- NP-completeness was discovered in the early 70's by Cook and Levin

Def: A language B is NP-**complete** if

1. $B \in NP$, and
2. $A \in NP \implies A$ is polynomial-time reducible to B

If this is true, we say $B \in NPC$ or $NP - C$.

- "A being in NP is sufficient for A to be polynomial-time reducible to B"

Satisfiability / SAT

- One of the first problems ever found $\in NPC$ is SAT
- A **Boolean formula** is an expression involving Boolean variables (T/F) and operations (EG $\neg, \wedge, \vee$)
  - Ex: $\phi = (\neg x \wedge y) \vee (x \wedge \neg z)$
- A Boolean formula is **satisfiable** iff there exists some set of variable values for which it evaluates to true

Def: The **satisfiability problem** SAT is the language

$$SAT = \{\langle \phi \rangle : \phi \texttt{ is a satisfiable Boolean formula}\}$$

- Clearly, we can decide $\phi \in SAT$ in time $2^{|\phi|}$ by brute force
- Also clearly, if we had some series of variable values $x_1, x_2, \ldots, x_i$ we could check it in polynomial (namely, linear) time ($SAT \in NP$)

Cook-Levin theorem (SAT is NP-complete)

- We know Boolean systems can simulate given TMs: This is what all computers are
- Given an input w to TM M, we can construct a Boolean expression that is true iff M accepts w
- All problems in NP are solvable by an NTM in polynomial time (by definition)
- Therefore, we can encode any NTM and input into an instance of SAT
- If SAT can be decided in deterministic polynomial time, we can determine **any NTM** in polynomial time! Therefore, $SAT \in P$ **iff** $P = NP$

Thm: Cook-Leven theorem. $SAT \in P$ iff $P = NP$.

Next up: Linear Bounded Automata (LBA)