The Concept of Limited Adversaries

Ahto Buldas

Feb 21, 2020

Computability

A function $A \xrightarrow{f} B$ is *computable* if:

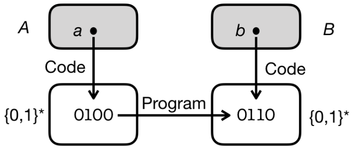
- The elements of sets A and B are suitably *encoded*, and
- There exists a program (finite sequence of commands) that transforms the code $\mathrm{Code}(a)$ of any $a \in A$ to the code $\mathrm{Code}(b)$ of $b = f(a) \in B$.

Code set: $\{0,1\}^*$ the set of all finite binary sequences:

$$\{0,1\}^* = \{0,1\}^0 \cup \{0,1\}^1 \cup \ldots \cup \{0,1\}^k \cup \ldots,$$

where $\{0,1\}^k$ is the set of all k-element binary sequences.

$\{0,1\}^0 = \{\sqcup\}$, where $\sqcup$ is the empty bitstring (of length 0).



Non-Computable Functions

Countable set A: there is a bijective (one-to-one and onto) function $\mathbb{N} \to A$, i.e. the elements of A can be enumerated with natural numbers

Not all functions are computable, because, for example, if $A = B = \mathbb{N}$:

- the set $\mathbb{N}^{\mathbb{N}}$ of all functions $\mathbb{N} \to \mathbb{N}$ is not countable
- the set $P \subset \{0,1\}^*$ of all finite programs is countable

Cantor's diagonal argument: For any enumerated set of functions $f_0, f_1, f_2, \ldots, f_k, \ldots$ of type $\mathbb{N} \to \mathbb{N}$, there is a function g that does not belong to the set. For example:

$$g(n) = f_n(n) + 1 \ .$$

Indeed, if $g = f_k$, then $f_k(k) = g(k) = f_k(k) + 1$, a contradiction.

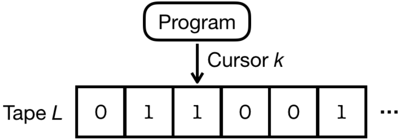
There are meaningful and useful functions that are not computable.

Turing Machine

Mathematical model of computation proposed by Alan Turing (1912–1954).

Turing machine has the following components:

- *Tape*: Infinite memory $L = (\ell_0, \ell_1, \ell_2, \ldots)$ with cells $\ell_i \in \{0, 1, \sqcup\}$.
- *Cursor* $k \in \mathbb{N}$: only the cell ℓ_k is accessible. Computational steps can increment or decrement k, or do nothing with it. Initially, $k = 0$.
- *Program*: Finite set S of states $s \in S$, where s_0 is the initial state and h (halt) is the final state.



Program of a Turing Machine

The next state s', new content ℓ'_k of the tape and the new cursor position k' is computed by the functions:

$$\begin{aligned} s' &:= \delta_s(s, \ell_k) \in S \\ \ell'_k &:= \delta_\ell(s, \ell_k) \in \{0, 1, \sqcup\} \\ k' &:= k + \delta_k(s, \ell_k) \in \{k-1, k, k+1\}\ , \text{ i.e. } \delta_k(s, \ell_k) \in \{-1, 0, +1\}. \end{aligned}$$

The initial state of the tape is considered to be the *input* and the final state as *output*.

For example, a function $\mathbb{N} \xrightarrow{f} \mathbb{N}$ is computable if there exists a Turing machine that transforms the initial state of the tape (if it represents an input x) to the code of $y = f(x)$ which must be on the tape when the machine reaches the end-state h.

Zero-Function is Computable

For example, the zero function $f(x) = 0, \forall x \in \mathbb{N}$ is computable because we have the following Turing machine that computes it:

s	ℓ_k	s'	ℓ'_k	$(k' - k)$
s_0	0	s_1	0	$+1$
	1	s_1	0	$+1$
	ϵ	h	0	0
s_1	0	s_1	$\sqcup$	$+1$
	1	s_1	$\sqcup$	$+1$
	ϵ	h	$\sqcup$	0

Here we assume that initially there is the binary code of x on the tape ending with empty cell $\sqcup$ and 0 is encoded by the tape $0\sqcup\sqcup\ldots$

Two Exercises

Ex 1: Find a Turing machine that computes the function $y = 2x + 1$ assuming that $x = b_0 2^0 + b_1 2^1 + \ldots + b_n 2^n$ (where $b_i \in \{0, 1\}$) is encoded by the tape $b_n b_{n-1} \ldots b_1 b_0 \sqcup \sqcup \ldots$

Ex 2: The same as in Ex 1, but use the opposite order encoding $b_0 b_1 \ldots b_{n-1} b_n \sqcup \sqcup \ldots$

Turing's Thesis

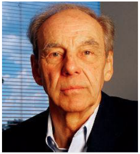
Though, Turing machine is a seemingly simple device, it is belived to be a universal model of computations.

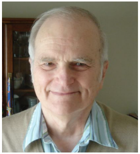
Turing's thesis: Everything that can be computed, can be computed with a Turing machine.

This is not a mathematical statement because "everything that can be computed" is not a precise mathematical term.

Measures of Computational Complexity

Juris Hartmanis (1928–) and Richard Stearns (1936–) started systematic studies in computational complexity





Running time: The number of state transitions before reaching the halt state h.

Memory: The number of memory cells used during the computation.

Program size: The number $|S|$ of states.

Bachmann–Landau Notations: Big O

Proposed by Paul Bachmann (1837–1920) and Edmund Landau (1877–1938)

Let f and g be real-valued functions of type $\mathbb{N} \to \mathbb{R}$

$f(n) = O(g(n))$: There exists $c \in \mathbb{R}$ and $n_0 \in \mathbb{N}$ so that for every $n \geq n_0$:

$$f(n) \leq c \cdot g(n) \ , \qquad \text{or equivalently} \qquad \frac{f(n)}{g(n)} \leq c \ .$$

$f(n) = \Omega(g(n))$ (Omega): iff $g(n) = O(f(n))$.

$f(n) = \Theta(g(n))$ (Theta): iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Bachmann–Landau Notations: Little o

$f(n) = o(g(n))$: iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, i.e. for every $\epsilon > 0$ there exists n_0 such that for every $n \geq n_0$:

$$f(n) \leq \epsilon \cdot g(n) \ , \qquad \text{or equivalently} \qquad \frac{f(n)}{g(n)} \leq \epsilon \ .$$

$f(n) = \omega(g(n))$: iff $g(n) = o(f(n))$.

$f(n) \sim g(n)$: iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$.

Combinatorial Problems

Decision Problem Instance: Given a description of a function f, decide whether there is ξ such that $f(\xi) = 0$.

Search Problem Instance: Given a description of a function f, find ξ such that $f(\xi) = 0$.

Decision Problem: A collection of decision problem instances of certain type.

Search Problem: A collection of search problem instances of certain type.

Primeness and Factoring

Primeness: Decide whether a given number $n \in \mathbb{N}$ is prime.

The corresponding function f_n:

$$f_n(\xi) = \begin{cases} 0 & \text{If } 1 < \gcd(\xi, n) < n \\ 1 & \text{otherwise} \end{cases}$$

Factoring: Given a composite number $n \in \mathbb{N}$, find a non-trivial divisor ξ.

The corresponding funcion is the same f_n.

Primeness as a decision problem: The collection of the descriptions of all functions $f_n(\xi)$ with any $n \in \mathbb{N}$.

n *is prime* if and only if $\forall \xi \colon f_n(\xi) = 1$

Decision Problems and Languages

Definition (Language)

A language $L \subseteq \{0,1\}^*$ is any set of finite bit-strings.

Example: The language PRIMES consists of all bit-strings x that are binary representations of prime numbers n.

Language recognition problem: Given a bit-string x, decide whether $x \in L$.

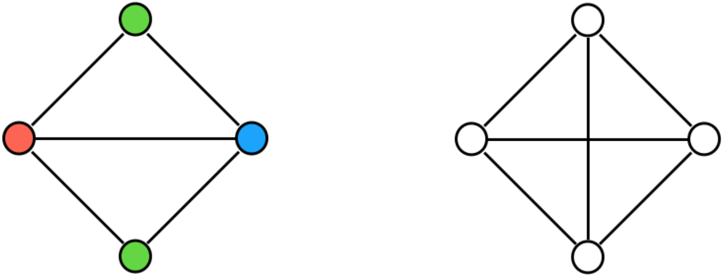
Every decision problem is equivalent to a language recognition problem!

Primeness as a language recognition problem: Given a bit-string x, decide if $x \in$ PRIMES.

3-Colouring

Definition (3-Colouring problem)

Given a graph decide whether the vertices can be coloured in a way that no two adjacent vertices are of the same color.



Boolean Satisfiability (SAT)

Definition (Boolean satisfiability(SAT) problem)

Given a Boolean formula, decide whether the atomic variables can be replaced with True and False so that the formula evaluates to True.

Definition (SAT language)

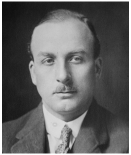
Given a coding rule Code, the set of all finite bitstrings c, for which there is a satisfiable Boolean formula φ, such that $c = \text{Code}(\varphi)$.

Definition (TAUTOLOGY language)

Given a coding rule Code, the set of all finite bitstrings c, for which there is a Boolean tautology φ, such that $c = \text{Code}(\varphi)$.

Class $\mathbf{P}$ and Cobham–Edmonds Thesis

Alan Cobham (1894–1973) and Jack Edmonds (1934–) were the first who defined feasible computations as polynomial-time computations.



Definition (Class $\mathbf{P}$)

A language L belongs to class $\mathbf{P}$ if there is a Turing machine V (the *verifier*) and a function $t(n) = n^{O(1)}$, such that or every $x \in \{0,1\}^*$:

- $x \in L$ iff V(x) outputs 1
- V(x) runs in time $t(|x|)$

Cobham–Edmonds thesis: computational problems can be feasibly solved on computational devices only if they lie in the complexity class $\mathbf{P}$.

Class $\mathbf{NP}$

Definition (Class $\mathbf{NP}$)

A language L belongs to class $\mathbf{NP}$ if there is a Turing machine V (the *verifier*) and a function $t(n) = n^{O(1)}$, such that or every $x \in \{0,1\}^*$:

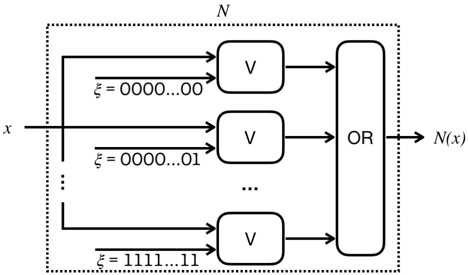
- $x \in L$ iff there is $\xi \in \{0,1\}^{t(|x|)}$ (a *certificate*) so that $1 \leftarrow \mathsf{V}(x,\xi)$
- $\mathsf{V}(x,\xi)$ runs in time $t(|x|)$

Example: SAT is in $\mathbf{NP}$: the verifier $\mathsf{V}(x,\xi)$ computes the value of the Boolean function x given a valuation ξ of its atomic variables.

Non-Deterministic Turing Machine (NDTM)

Definition (Non-Deterministic Turing Machine with running time t)

A machine N that uses an ordinary Turing machine V so that for any input $x \in \{0,1\}^*$ the machine executes $y_\xi \leftarrow \mathsf{V}(x,\xi)$ for all $\xi \in \{0,1\}^{t(|x|)}$. If there is ξ so that $y_\xi = 1$, then $1 \leftarrow N(x)$, otherwise $0 \leftarrow N(x)$.



$\mathbf{NP}$ = languages recognizable by poly-time NDTMs

Class $\mathbf{coNP}$

Definition (Class $\mathbf{coNP}$)

A language L belongs to class $\mathbf{coNP}$ if there is a Turing machine V (the *verifier*) and a function $t(n) = n^{O(1)}$, such that or every $x \in \{0,1\}^*$:

- $x \in L$ iff $1 \leftarrow \mathsf{V}(x, \xi)$ for every $\xi \in \{0,1\}^{t(|x|)}$
- $\mathsf{V}(x, \xi)$ runs in time $t(|x|)$

Example: TAUTOLOGY is in $\mathbf{coNP}$: the verifier $\mathsf{V}(x, \xi)$ computes the value of the Boolean function x given a valuation ξ of its atomic variables.

Exercise: Show that any language L is in $\mathbf{coNP}$ if and only if its complement $\overline{L} = \{x \in \{0,1\}^* \colon x \notin L\}$ is in $\mathbf{NP}$.

Non-Uniform Computations and the Class $\mathbf{P}/\mathbf{poly}$

Definition (Class $\mathbf{P}/\mathbf{poly}$)

A language L belongs to class $\mathbf{P}/\mathbf{poly}$ if there is a Turing machine V (the *verifier*), a function $t(n) = n^{O(1)}$, and a sequence $(\xi_0, \xi_1, \xi_2, \ldots)$ of *advice strings* with size $|\xi_n| \in \{0,1\}^{t(n)}$, such that or every $x \in \{0,1\}^*$:

- $x \in L$ iff $1 \leftarrow \mathsf{V}(x, \xi_{|x|})$
- $\mathsf{V}(x, \xi_{|x|})$ runs in time $t(|x|)$

Randomized Computations

Randomized TM: Uses additional input for a random string ω, i.e. the computation is $y \leftarrow \mathsf{M}(\omega, x)$, where $\omega \leftarrow \{0,1\}^t$ is a uniformly chosen random string, where t (the number of random bits) is $t \geq T(\mathsf{M}, x)$, where $T(\mathsf{M}, x)$ is the worst-case running time of M.

Class $\mathbf{RP}$ and Monte Carlo Algorithms

Definition (Class $\mathbf{RP}$)

A language L belongs to class $\mathbf{RP}$ if there is a Turing machine M_1 and a function $t(n) = n^{O(1)}$, such that or every $x \in \{0,1\}^*$:

- If $x \in L$ then $\mathsf{P}_\xi[1 \leftarrow \mathsf{M}_1(x,\xi)] > \frac{1}{2}$ where $\xi \leftarrow \{0,1\}^{t(|x|)}$ is chosen uniformly at random
- If $x \notin L$ then $\mathsf{P}_\xi[1 \leftarrow \mathsf{M}_1(x,\xi)] = 0$
- $\mathsf{M}_1(x,\xi)$ runs in time $t(|x|)$

Monte-Carlo algorithm: Given x, run $\mathsf{M}_1(x,\xi)$ with m independent values of ξ. If $1 \leftarrow \mathsf{M}_1(x,\xi)$ for some ξ, return 1, otherwise return 0.

If $x \in L$, then the Monte-Carlo algorithm returns 0 with probability $< \frac{1}{2^m}$.

- If the algorithm returns 1, then we know that $x \in L$.
- If the algorithm returns 0, then $x \notin L$ with probability $1 - \frac{1}{2^m}$.

Class coRP and Monte Carlo Algorithms

Definition (Class coRP)

A language L belongs to class $\mathbf{coRP}$ if there is a Turing machine M_0 and a function $t(n) = n^{O(1)}$, such that or every $x \in \{0,1\}^*$:

- If $x \in L$ then $\mathsf{P}_\xi[1 \leftarrow \mathsf{M}_0(x,\xi)] = 1$ where $\xi \leftarrow \{0,1\}^{t(|x|)}$ is chosen uniformly at random
- If $x \notin L$ then $\mathsf{P}_\xi[1 \leftarrow \mathsf{M}_0(x,\xi)] < \frac{1}{2}$
- $\mathsf{M}_0(x,\xi)$ runs in time $t(|x|)$

Monte-Carlo algorithm: Given x, run $\mathsf{M}_0(x,\xi)$ with m independent values of ξ. If $1 \leftarrow \mathsf{M}_0(x,\xi)$ for some ξ, return 1, otherwise return 0.

If $x \notin L$, then the Monte-Carlo algorithm returns 1 with probability $< \frac{1}{2^m}$.

- If the algorithm returns 0, then we know that $x \notin L$.
- If the algorithm returns 1, then $x \in L$ with probability $1 - \frac{1}{2^m}$.

Class $\mathbf{ZPP}$ and Las Vegas Algorithms

Definition (Class $\mathbf{ZPP}$)

$\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$

Las Vegas algorithm: Given x, run $\mathsf{M}_1(x, \xi)$ and $\mathsf{M}_0(x, \xi)$ with independently chosen ξ until $1 \leftarrow \mathsf{M}_1(x, \xi)$ or $0 \leftarrow \mathsf{M}_0(x, \xi)$.

- If $x \in L$, the Las Vegas algorithm returns 1
- If $x \notin L$, the Las Vegas algorithm returns 0
- The average running time is $4t(|x|)$

Class $\mathbf{BPP}$ and Majority Voting

Definition (Class $\mathbf{BPP}$)

A language L belongs to class $\mathbf{BPP}$ if there is a Turing machine M and a function $t(n) = n^{O(1)}$, such that or every $x \in \{0,1\}^*$:

- If $x \in L$ then $\mathsf{P}_\xi[1 \leftarrow \mathsf{M}(x,\xi)] > \frac{3}{4}$ where $\xi \leftarrow \{0,1\}^{t(|x|)}$ is chosen uniformly at random
- If $x \notin L$ then $\mathsf{P}_\xi[1 \leftarrow \mathsf{M}(x,\xi)] < \frac{1}{4}$
- $\mathsf{M}(x,\xi)$ runs in time $t(|x|)$

Majority Voting: Given x, run $\mathsf{M}(x,\xi)$ with m independent values of ξ. If more than $\frac{m}{2}$ outputs were 1, return 1, otherwise return 0.

How large should m be?: $\rightsquigarrow$ Chernoff-Hoeffding bounds.

Chernoff-Hoeffding Bounds

Herman Chernoff (1923–) and Wassily Hoeffding (1914–1991) proved bounds on tail distributions of sums of independent random variables.

Theorem: Let $x_1, \ldots, x_m$ be independent identically distributed $0/1$ random variables, $p = \mathsf{P}[x_i = 1]$ and $X = \sum_{i=1}^{m} x_i$. Then for any $0 \leq \Theta \leq 1$:

$$\mathsf{P}[X \geq (1+\Theta)pm] \leq e^{-\frac{\Theta^2}{3}pm} \quad (1)$$

$$\mathsf{P}[X \leq (1-\Theta)pm] \leq e^{-\frac{\Theta^2}{2}pm} . \quad (2)$$

The proof is based on two lemmas:

Lemma 1: If $0 \leq \Theta \leq 1$ then $-\frac{\Theta^2}{2} \leq \Theta - (1+\Theta)\ln(1+\Theta) \leq -\frac{\Theta^2}{3}$.

Lemma 2: If $0 \leq \Theta \leq 1$ then $\Theta - (1-\Theta)\ln(1-\Theta) \geq \frac{\Theta^2}{2}$.

Proof of the First Chernoff-Hoeffding Bound (1)

$\mathsf{P}[X \geq (1+\Theta)pm] = \mathsf{P}[e^{tX} \geq e^{t(1+\Theta)pm}]$ for any $0 < t$.

By Markov's inequality: $\mathsf{P}[e^{tX} \geq k \cdot \mathbf{E}[e^{tX}]] \leq 1/k$ for any $k > 0$.

We take $k = e^{t(1+\Theta)pm}(\mathbf{E}[e^{tX}])^{-1}$. Then

$$\mathsf{P}[X \geq (1+\Theta)pm] \leq e^{-t(1+\Theta)pm}\,\mathbf{E}[e^{tX}] \ ,$$

and as $\mathbf{E}[e^{tX}] = (\mathbf{E}[e^{tx_1}])^m = (1 + p(e^t - 1))^m$, we have:

$$\begin{aligned} \mathsf{P}[X \geq (1+\Theta)pm] &\leq e^{-t(1+\Theta)pm}(1+p(e^t-1))^m \\ &\leq e^{-t(1+\Theta)pm} \cdot e^{pm(e^t-1)} \ . \end{aligned}$$

This holds because $1 + a \leq e^a$ for every $a > 0$. In our case $a = p(e^t - 1)$.

Finally, by taking $t = \ln(1+\Theta)$, we obtain from Lemma 1 that

$$\mathsf{P}[X \geq (1+\Theta)pm] \leq e^{pm[\Theta-(1+\Theta)\ln(1+\Theta)]} \leq e^{-\frac{\Theta^2}{3}pm} \ .$$

Proof of the Second Chernoff-Hoeffding Bound (2)

$\mathsf{P}[X \leq (1-\Theta)pm] = \mathsf{P}[pm - X \geq \Theta pm] = \mathsf{P}[e^{t(pm-X)} \geq e^{t\Theta pm}]$ for any $0 < t$.

By Markov's inequality: $\mathsf{P}[e^{t(pm-X)} \geq k \cdot \mathbf{E}[e^{t(pm-X)}]] \leq \frac{1}{k}$ for any $k > 0$.

We take $k = e^{t\Theta pm}(\mathbf{E}[e^{t(pm-X)}])^{-1}$. Then

$$\mathsf{P}[X \leq (1-\Theta)pm] \leq e^{-t\Theta pm} \cdot \mathbf{E}[e^{t(pm-X)}] = e^{t(1-\Theta)pm} \cdot \mathbf{E}[e^{-tX}] \ ,$$

and as $\mathbf{E}[e^{-tX}] = (\mathbf{E}[e^{-tx_1}])^m = (1-p(1-e^{-t}))^m$, we have:

$$\begin{aligned} \mathsf{P}[X \leq (1-\Theta)pm] &\leq e^{-t(1-\Theta)pm}(1-p(1-e^{-t}))^m \\ &\leq e^{-t(1-\Theta)pm} \cdot e^{-pm(1-e^{-t})} \\ &= e^{-pm[t(1-\Theta)+1-e^{-t}]} \ . \end{aligned}$$

Finally, by taking $t = -\ln(1-\Theta)$ we obtain from Lemma 2 that

$$\mathsf{P}[X \leq (1-\Theta)pm] \leq e^{-pm[\Theta-(1-\Theta)\ln(1-\Theta)]} \leq e^{-\frac{\Theta^2}{2}pm}.$$

Proof of Lemma 1

Lemma 1: If $0 \le \Theta \le 1$ then $-\frac{\Theta^2}{2} \le \Theta - (1+\Theta)\ln(1+\Theta) \le -\frac{\Theta^2}{3}$.

Proof: First, note that

$$\begin{aligned}\Theta - (1+\Theta)\ln(1+\Theta) &= \Theta - (1+\Theta)\cdot\left(\frac{\Theta}{1} - \frac{\Theta^2}{2} + \frac{\Theta^3}{3} - \frac{\Theta^4}{4} + \ldots\right) \\ &= -\frac{\Theta^2}{1\cdot 2} + \frac{\Theta^3}{2\cdot 3} - \frac{\Theta^4}{3\cdot 4} + \frac{\Theta^5}{4\cdot 5}\ldots \\ &= \sum_{n=2}^{\infty}(-1)^{n-1}\frac{\Theta^n}{n(n-1)} \quad .\end{aligned}$$

As the series $r = \frac{\Theta^3}{2\cdot 3} - \frac{\Theta^4}{3\cdot 4} + \frac{\Theta^5}{4\cdot 5}\ldots$ is with alternating signs and their absolute values are strongly decreasing, because $\frac{\Theta^n}{(n-1)n} \ge \frac{\Theta^{n+1}}{n(n+1)}$ directly follows from $\frac{n-1}{n+1}\Theta \le 1$. Hence, the sum of this series is positive, because the fist term is positive.

Proof of Lemma 1 continues

Consequently:

$$\Theta - (1+\Theta)\ln(1+\Theta) = -\frac{\Theta^2}{2} + r \geq -\frac{\Theta^2}{2} \ .$$

Analogously, we claim that the series $s = \frac{\Theta^4}{3\cdot 4} - \frac{\Theta^5}{4\cdot 5} + \frac{\Theta^6}{4\cdot 5} - \ldots$ has positive sum and hence:

$$\begin{aligned} \Theta - (1+\Theta)\ln(1+\Theta) &= -\frac{\Theta^2}{2} + \frac{\Theta^3}{6} - s \leq -\frac{\Theta^2}{2} + \frac{\Theta^3}{6} \leq -\frac{\Theta^2}{2} + \frac{\Theta^2}{6} \\ &= -\frac{\Theta^2}{3} \ . \end{aligned}$$

Proof of Lemma 2

Lemma 2: If $0 \leq \Theta \leq 1$ then $\Theta - (1-\Theta)\ln(1-\Theta) \geq \frac{\Theta^2}{2}$.

Proof: It is easy to see that

$$\Theta - (1-\Theta)\ln(1-\Theta) = \frac{\Theta^2}{2\cdot 1} + \frac{\Theta^3}{3\cdot 2} + \frac{\Theta^4}{4\cdot 3} + \ldots = \sum_{n=2}^{\infty} \frac{\Theta^n}{n(n-1)} \ ,$$

from which the inequality directly follows.

Analysis of the Voting Algorithm

For $i = 1 \dots m$ let $x_i \in \{0,1\}$ be the error variables, i.e. $x_i = 1$ iff the i-th sample b_i of $M(x)$ wrongly reflects the truth value of $x \in L$.

By the definition of $\mathbf{BPP}$, we have $p = \mathsf{P}[x_i = 1] \leq \frac{1}{4}$.

By taking $\Theta = 1$ in the first Chernoff-Hoeffding bound, we obtain

$$\mathsf{P}\left[\sum_{i=1}^{m} x_i \geq \frac{m}{2}\right] \leq e^{-\frac{m}{12}} \ .$$

Hence, the voting algorithm has error $< e^{-\frac{m}{12}}$.

For example, if the desired error is e^{-100}, it is sufficient to take $m = 1200$.

$\mathbf{BPP}_\epsilon$

Let $\epsilon\colon \mathbb{N} \to [0,1]$ be a function.

Definition (Class $\mathbf{BPP}_\epsilon$)

A language $L \subseteq \{0,1\}^*$ belongs to the class $\mathbf{BPP}_\epsilon$ if there is a poly-time probabilistic Turing machine N such that for every $x \in \{0,1\}^n$:

- $x \in L \;\Rightarrow\; \mathsf{P}[N(x) = 1] > 1 - \epsilon(|x|)$
- $x \notin L \;\Rightarrow\; \mathsf{P}[N(x) = 1] < \epsilon(|x|)$

Exercise: By using Chernoff bounds, prove the following:

- If $\epsilon(n) = 2^{-n^{O(1)}}$, then $\mathbf{BPP}_\epsilon = \mathbf{BPP}$
- If $\epsilon(n) = n^{-O(1)}$, then $\mathbf{BPP}_{\frac{1}{2}-\epsilon} = \mathbf{BPP}$

Karp Reductions

Defined by Richard Manning Karp (1935–).

Reduce one combinatorial problem to another.

Definition (Karp reduction)

A *Karp reduction* of a language L_1 to a language L_2 is a poly-time computable function $f\colon \{0,1\}^* \to \{0,1\}^*$, such that for every $x \in \{0,1\}^*$:

$$x \in L_1 \quad \Leftrightarrow \quad f(x) \in L_2 \ .$$

We write $L_1 \leq_{\mathrm{p}} L_2$ if there is a Karp reduction of L_1 to L_2.

Exercise 1: Show that if $L_1 \leq_{\mathrm{p}} L_2$ and $L_2 \leq_{\mathrm{p}} L_3$, then $L_1 \leq_{\mathrm{p}} L_3$.

Exercise 2: Show that if $L_2 \in \mathbf{P}$ and $L_1 \leq_{\mathrm{p}} L_2$, then $L_1 \in \mathbf{P}$.

Exercise 3: Show that if $L_2 \in \mathbf{BPP}$ and $L_1 \leq_{\mathrm{p}} L_2$, then $L_1 \in \mathbf{BPP}$.

NP-Completeness and Cook-Levin Theorem

Stephen Cook (1939–) and Leonid Levin (1948–) proved the existence of **NP**-complete problems.

Definition (**NP**-hardness, **NP**-completeness)

A language L is **NP**-*hard*, if $L' \leq_{\mathrm{p}} L$ for every $L' \in \mathbf{NP}$. If, in addition, $L \in \mathbf{NP}$, then L is said to be **NP**-*complete*.

Theorem (Cook, Levin, 1971)

Satisfiability (SAT) is **NP***-complete.*

Exercise 1: Show that if $L' \leq_{\mathrm{p}} L$ and L' is **NP**-complete, then L is **NP**-hard.

Exercise 2: Show that if SAT $\leq_{\mathrm{p}} L$ and $L \in \mathbf{NP}$, then L is **NP**-complete.

Other $\mathbf{NP}$-Complete Problems

In 1972, Richard Karp proved $\mathbf{NP}$-completeness of 21 combinatorial problems, including:

3-Colouring: Given a graph, decide whether the vertices can be coloured in a way that no two adjacent vertices are of the same color.

Subset sum: Given a set (or multiset) of integers, decide if there is a non-empty subset whose sum is zero.

Clique: Given a graph and an integer k, decide if there is a complete subgraph with k vertices.

$\mathbf{P}$ vs $\mathbf{NP}$: The Holy Grail of Computer Science

John Edward Hopcroft (1939–), after a fierce debate at the STOC 1971 conference, brought everyone to a consensus that $\mathbf{P} = \mathbf{NP}$ should be solved soon.

So far, it is one of the greatest unsolved problems of mathematics.

It is one of the seven *Millennium Prize Problems*: The Clay Mathematics Institute offers 1 million USD reward for proving or disproving $\mathbf{P} = \mathbf{NP}$.

Most computer scientists believe that $\mathbf{P} \neq \mathbf{NP}$.

Oracle Machines

Oracle is any function $\mathcal{O}\colon \{0,1\}^* \to \{0,1\}^*$, not necessarily computable.

Definition (Oracle Machine $M^{\mathcal{O}}$)

A Turing machine that, in addition to ordinary configuration, has:

- *oracle tape* with oracle cursor for read/write operations
- *oracle calls* (can be executed at any state): for any $x \in \{0,1\}^*$ written in the oracle tape, the contents of the oracle tape is instantly replaced with $\mathcal{O}(x)$

The number of oracle calls of $M^{\mathcal{O}}$ does not exceed the running time t.

Turing Reductions

Definition (Turing reduction)

A *Turing reduction* of a language L_1 to a language L_2 is a poly-time oracle machine $M^{\mathcal{O}}$ such that for every $x \in \{0,1\}^*$:

$$x \in L_1 \quad \Leftrightarrow \quad 1 \leftarrow M^{\mathcal{O}}(x) \ ,$$

where $\mathcal{O}$ is the characteristic function of L_2, i.e. for every z:

$$\mathcal{O}(z) = \begin{cases} 1 & \text{if } z \in L_2 \\ 0 & \text{if } z \notin L_2 \end{cases}$$

We write $L_1 \leq_{\mathrm{p}}^{\mathrm{T}} L_2$ if there is a Turing reduction of L_1 to L_2.

Security and Proofs of Security

Breaking a cryptosystem is solving an instance of a *search problem*.

Practically Secure Cryptosystem: Too costly to break.

Proof of Practical Security: If the cryptosystem can be broken with cost S, then a hard instance of a combinatorial problem can be solved with cost S'.

Polynomially Secure Cryptosystem: Any efficient (poly-time) adversary has negligible success probability.

Proof of Polynomial Security: A hard combinatorial problem can be Turing-reduced to the problem of breaking the cryptosystem.

Polynomial security is of limited practical relevance, because real-life cryptosystems tend to be fixed and finite.

Practical Measures of Computational Costs and Security

Intuition: A cryptosystem is S-secure, if it cannot be broken with cost less than S.

Engineers have to estimate the *total cost* of potential attacks, including:

- Algorithm development
- Coding
- Hardware (memory, processors, etc.)
- Energy

Total cost is computed from *technical complexity*, given the (monetary) prices of computational resources.

Technical complexity itself must be *price-independent*.

Time as a Measure of Technical Complexity

Computational time alone is not a good measure for technical complexity:

Theorem

For every function $f\colon \{0,1\}^n \to \{0,1\}^n$ (where n is constant) there is a Turing machine M *that computes the function in time $t = 2n$.*

Proof.

The machine M has $(n+1)2^n$ states: a tree like structure of $2^n - 1$ of states to encode the input x into one of 2^n possible input value states. Then for every such state we have a sequence of n states to write out the output $f(x) \in \{0,1\}^n$, and the halt state. The machine needs n steps to determine the input state and n steps to write out the output. □

By such definition, *no fixed one-way functions exist!*

Time + Code Size

Much more relevant complexity measure.

Can be converted to pure time-measure.

Assumption: adversaries have to load their program before the attack.

Under such assumption, program size converts to computational time.

Time-Success Ratio

Attacks may succeed with certain probability.

Definition (S-security)

A primitive is *S-secure* if every adversary with running time t has success probability $\delta \leq \frac{t}{S}$.

Equivalently:

Definition (S-security)

A primitive is *S-secure* if every adversary has time-success ratio $\frac{t}{\delta} \geq S$.

Time-Success Ratio: Motivation

Time-success ratio $\frac{t}{\delta}$ is a natural measure of technical complexity.

Consider a there is a prize of P monetary units offered for breaking a cryptosystem.

You know an attack A with running time t and success probability δ.

Under which conditions it is economically beneficial to take the challenge?

Let α denote the total cost of one computational step. Then:

- the cost of the attack is αt
- the average income is δP

Hence, the attack is beneficial if $\delta P - \alpha t > 0$, i.e. if $\frac{t}{\delta} < \frac{P}{\alpha}$, where $\frac{P}{\alpha}$ is the prize expressed in computational-step units.

Hence, $\frac{t}{\delta}$ measures the cost in computational steps.

Security Bits

Definition (Security bits)

A primitive has k *bits of security* iff it is S-secure, where $\log_2 S \geq k$.

Usually, the time t is measured in *block-cipher units*.

1 block-cipher unit $=$ time needed for the encryption of one block of data with an ordinary block-cipher, or computing a hash of one block of data.

Example: One-Way Functions

Let $f\colon X \to Y$ be a function.

Adversary is a probabilistic Turing Machine A that participates in the following attack scenario:

1. An input $x \leftarrow X$ is chosen randomly.
2. The output $y = f(x)$ is computed.
3. Given y as input, the adversary A computes $x' \leftarrow A(y)$.
4. Adversary is successful iff $f(x') = y$.

Definition (S-secure One-Way Function)

A function f is *S-secure one-way* if every adversary A has time-success ratio $\frac{t}{\delta} \geq S$.