Theory of Computation Lecture Notes

Prof. Yuh-Dauh Lyuu
Dept. Computer Science & Information Engineering
and
Department of Finance
National Taiwan University

Class Information

- Papadimitriou. *Computational Complexity*. 2nd printing. Addison-Wesley. 1995.
- Check

  `www.csie.ntu.edu.tw/~lyuu/complexity/2006`

  for lecture notes.

Problems and Algorithms

I have never done anything "useful."
— Godfrey Harold Hardy (1877–1947),
A Mathematician's Apology (1940)

What This Course Is All About

Computability: What can be computed?

- What is computation anyway?
- There are *well-defined* problems that cannot be computed.
- In fact, "most" problems cannot be computed.

What This Course Is All About (concluded)

Complexity: What is a computable problem's inherent complexity?

- Some computable problems require at least exponential time and/or space; they are **intractable**.
  - Can't you let the Moore law take care of it?[a]
- Some practical problems require superpolynomial resources unless certain conjectures are disproved.
- Other resource limits besides time and space?
  - Program size, circuit size (growth), number of random bits, etc.

[a] Contributed by Ms. Amy Liu (`J94922016`) on May 15, 2006.

Tractability and intractability

- Polynomial in terms of the input size n defines tractability.
  - n, $n \log n$, n^2, n^{90}.
  - Time, space, circuit size, number of random bits, etc.
- It results in a fruitful and practical theory of complexity.
- Few practical, tractable problems require a large degree.
- Exponential-time or superpolynomial-time algorithms are usually impractical.
  - $n^{\log n}$, $2^{\sqrt{n}}$, 2^n, $n! \sim \sqrt{2\pi n}\,(n/e)^n$.

Growth of Factorials

n	$n!$	n	$n!$
1	1	9	362,880
2	2	10	3,628,800
3	6	11	39,916,800
4	24	12	479,001,600
5	120	13	6,227,020,800
6	720	14	87,178,291,200
7	5040	15	1,307,674,368,000
8	40320	16	20,922,789,888,000

Turing Machines

What Is Computation?

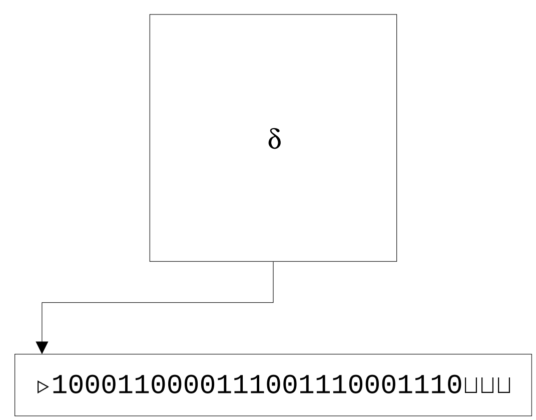
- That can be coded in an **algorithm**.
- An algorithm is a detailed step-by-step method for solving a problem.
  - The Euclidean algorithm for the greatest common divisor is an algorithm.
  - "Let s be the least upper bound of compact set A" is not an algorithm.
  - "Let s be a smallest element of a finite-sized array" can be solved by an algorithm.

Turing Machines[a]

- A Turing machine (TM) is a quadruple $M = (K, \Sigma, \delta, s)$.
- K is a finite set of **states**.
- $s \in K$ is the **initial state**.
- Σ is a finite set of **symbols** (disjoint from K).
  - Σ includes $\bigsqcup$ (blank) and $\triangleright$ (first symbol).
- $\delta : K \times \Sigma \to (K \cup \{h, \text{"yes"}, \text{"no"}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$ is a **transition function**.
  - $\leftarrow$ (left), $\rightarrow$ (right), and $-$ (stay) signify cursor movements.

[a] Turing (1936).

A TM Schema



"Physical" Interpretations

- The tape: computer memory and registers.
- δ: program.
- K: instruction numbers.
- s: "`main()`" in C.
- Σ: **alphabet** much like the ASCII code.

More about δ

- The program has the **halting state** (h), the **accepting state** ("yes"), and the **rejecting state** ("no").
- Given current state $q \in K$ and current symbol $\sigma \in \Sigma$,

$$\delta(q, \sigma) = (p, \rho, D).$$

  - It specifies the next state p, the symbol ρ to be written over σ, and the direction D the cursor will move *afterwards*.
- We require $\delta(q, \rhd) = (p, \rhd, \rightarrow)$ so that the cursor never falls off the left end of the string.

The Operations of TMs

- Initially the state is s.
- The string on the tape is initialized to a $\rhd$, followed by a *finite-length* string $x \in (\Sigma - \{\sqcup\})^*$.
- x is the **input** of the TM.
  - The input must not contain $\sqcup$s (why?)!
- The cursor is pointing to the first symbol, always a $\rhd$.
- The TM takes each step according to δ.
- The cursor may overwrite $\sqcup$ to make the string longer during the computation.

The Halting of a TM

- A TM M may **halt** in three cases.

  "yes": M **accepts** its input x, and $M(x) =$ "yes".

  "no": M **rejects** its input x, and $M(x) =$ "no".

  h**:** $M(x) = y$, where the string consists of a $\rhd$, followed by a finite string y, whose last symbol is not $\sqcup$, followed by a string of $\sqcup$s.
  - y is the **output** of the computation.
  - y may be empty denoted by ϵ.
- If M never halts on x, then write $M(x) = \nearrow$.

Why TMs?

- Because of the simplicity of the TM, the model has the advantage when it comes to complexity issues.
- One can develop a complexity theory based on C++ or Java, say.
- But the added complexity does not yield additional fundamental insights.
- We will describe TMs in pseudocode.

Remarks

- A problem is computable if there is a TM that halts with the correct answer.
  - If a TM (i.e., program) does not always halt, it does not solve the problem, assuming the problem is computable.[a]
  - OS does not halt as it does not solve a well-defined problem (but parts of it do).[b]

[a] Contributed by Ms. Amy Liu (J94922016) on May 15, 2006. Control-C is not a legitimate way to halt a program.

[b] Contributed by Mr. Shuai-Peng Huang (J94922019) on May 15, 2006.

Remarks (concluded)

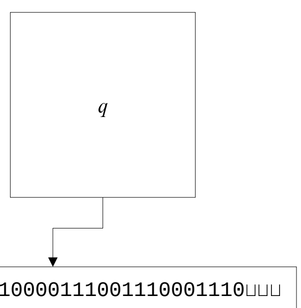
- Any computation model must be physically realizable.
  - A model that requires nearly infinite precision to build is not physically realizable.
  - For example, if the TM required a voltage of 100 ± 10^{-100} to work, it would not be considered a successful model for computation.

The Concept of Configuration

- A **configuration** is a complete description of the current state of the computation.
- The specification of a configuration is sufficient for the computation to continue as if it had not been stopped.
  - What does your PC save before it sleeps?
  - Enough for it to resume work later.

Configurations (concluded)

- A configuration is a triple (q, w, u):
  - $q \in K$.
  - $w \in \Sigma^*$ is the string to the left of the cursor (inclusive).
  - $u \in \Sigma^*$ is the string to the right of the cursor.
- Note that (w, u) describes both the string and the cursor position.



- $w = \triangleright$1000110000.
- $u =$ 111001110001110.

Yielding

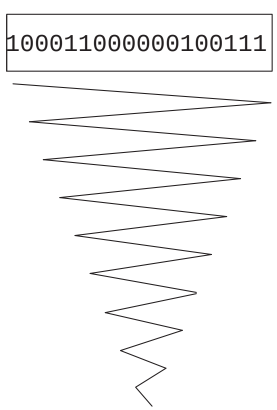
- Fix a TM M.
- Configuration (q, w, u) **yields** configuration (q', w', u') in one step,
$$(q, w, u) \xrightarrow{M} (q', w', u'),$$
if a step of M from configuration (q, w, u) results in configuration (q', w', u').
- $(q, w, u) \xrightarrow{M^k} (q', w', u')$: Configuration (q, w, u) yields configuration (q', w', u') in $k \in \mathbb{N}$ steps.
- $(q, w, u) \xrightarrow{M^*} (q', w', u')$: Configuration (q, w, u) yields configuration (q', w', u').

Example: How to Insert a Symbol

- We want to compute $f(x) = ax$.
  - The TM moves the last symbol of x to the right by one position.
  - It then moves the next to last symbol to the right, and so on.
  - The TM finally writes a in the first position.
- The total number of steps is $O(n)$, where n is the length of x.

Palindromes

- A string is a **palindrome** if it reads the same forwards and backwards (e.g., `001100`).
- A TM program can be written to recognize palindromes:
  - It matches the first character with the last character.
  - It matches the second character with the next to last character, etc. (see next page).
  - "yes" for palindromes and "no" for nonpalindromes.
- This program takes $O(n^2)$ steps.
- Can we do better?



Decidability and Recursive Languages

- Let $L \subseteq (\Sigma - \{\sqcup\})^*$ be a **language**, i.e., a set of strings of symbols with a finite length.
  - For example, $\{0, 01, 10, 210, 1010, \ldots\}$.
- Let M be a TM such that for any string x:
  - If $x \in L$, then $M(x) =$ "yes."
  - If $x \notin L$, then $M(x) =$ "no."
- We say M **decides** L.
- If L is decided by some TM, then L is **recursive**.
  - Palindromes over $\{0, 1\}^*$ are recursive.

Acceptability and Recursively Enumerable Languages

- Let $L \subseteq (\Sigma - \{\sqcup\})^*$ be a language.
- Let M be a TM such that for any string x:
  - If $x \in L$, then $M(x) =$ "yes."
  - If $x \notin L$, then $M(x) = \nearrow$.
- We say M **accepts** L.

Acceptability and Recursively Enumerable Languages (concluded)

- If L is accepted by some TM, then L is a **recursively enumerable language**.
  - A recursively enumerable language can be generated by a TM, thus the name.
  - That is, there is an algorithm such that for every $x \in L$, it will be printed out eventually.

Recursive and Recursively Enumerable Languages

Proposition 1 *If L is recursive, then it is recursively enumerable.*

- We need to design a TM that accepts L.
- Let TM M decide L.
- We next modify M's program to obtain M' that accepts L.
- M' is identical to M except that when M is about to halt with a "no" state, M' goes into an infinite loop.
- M' accepts L.

Turing-Computable Functions

- Let $f : (\Sigma - \{\sqcup\})^* \to \Sigma^*$.
  - Optimization problems, root finding problems, etc.
- Let M be a TM with alphabet Σ.
- M **computes** f if for any string $x \in (\Sigma - \{\sqcup\})^*$, $M(x) = f(x)$.
- We call f a **recursive function**[a] if such an M exists.

[a] Gödel (1931).

Church's Thesis or the Church-Turing Thesis

- What is computable is Turing-computable; TMs are algorithms (Kleene 1953).
- Many other computation models have been proposed.
  - Recursive function (Gödel), λ calculus (Church), formal language (Post), assembly language-like RAM (Shepherdson & Sturgis), boolean circuits (Shannon), extensions of the Turing machine (more strings, two-dimensional strings, and so on), etc.
- All have been proved to be equivalent.
- No "intuitively computable" problems have been shown not to be Turing-computable (yet).

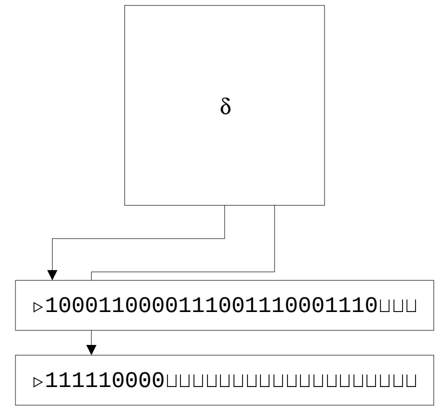
Extended Church's Thesis

- All "reasonably succinct encodings" of problems are *polynomially related*.
  - Representations of a graph as an adjacency matrix and as a linked list are both succinct.
  - The *unary* representation of numbers is not succinct.
  - The *binary* representation of numbers is succinct.
    - 1001 vs. 111111111.
- All numbers for TMs will be binary from now on.

Turing Machines with Multiple Strings

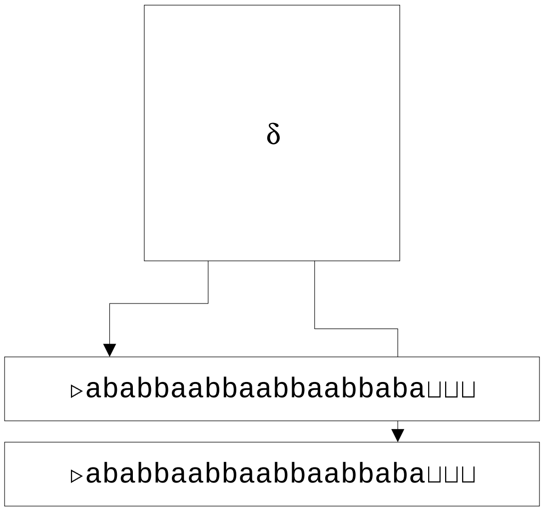
- A k-string Turing machine (TM) is a quadruple $M = (K, \Sigma, \delta, s)$.
- K, Σ, s are as before.
- $\delta : K \times \Sigma^k \to (K \cup \{h, \text{"yes"}, \text{"no"}\}) \times (\Sigma \times \{\leftarrow, \rightarrow, -\})^k$.
- All strings start with a $\triangleright$.
- The first string contains the input.
- Decidability and acceptability are the same as before.
- When TMs compute functions, the output is on the last (kth) string.

A 2-String TM



PALINDROME Revisited

- A 2-string TM can decide PALINDROME in $O(n)$ steps.
  - It copies the input to the second string.
  - The cursor of the first string is positioned at the first symbol of the input.
  - The cursor of the second string is positioned at the last symbol of the input.
  - The two cursors are then moved in opposite directions until the ends are reached.
  - The machine accepts if and only if the symbols under the two cursors are identical at all steps.



Configurations and Yielding

- The concept of configuration and yielding is the same as before except that a configuration is a $(2k+1)$-triple

$$(q, w_1, u_1, w_2, u_2, \ldots, w_k, u_k).$$

  - $w_i u_i$ is the ith string.
  - The ith cursor is reading the last symbol of w_i.
  - Recall that $\rhd$ is each w_i's first symbol.
- The k-string TM's initial configuration is

$$(s, \overbrace{\rhd, x, \rhd, \epsilon, \rhd, \epsilon, \ldots, \rhd, \epsilon}^{2k}).$$

Time Complexity

- The multistring TM is the basis of our notion of the time expended by TM computations.
- If for a k-string TM M and input x, the TM halts after t steps, then the **time required by M on input x** is t.
- If $M(x) = \nearrow$, then the time required by M on x is ∞.
- Machine M **operates within time** $f(n)$ for $f : \mathbb{N} \to \mathbb{N}$ if for any input string x, the time required by M on x is at most $f(\mid x \mid)$.
  - $\mid x \mid$ is the length of string x.
  - Function $f(n)$ is a **time bound** for M.

Time Complexity Classes[a]

- Suppose language $L \subseteq (\Sigma - \{\bigsqcup\})^*$ is decided by a multistring TM operating in time $f(n)$.
- We say $L \in \text{TIME}(f(n))$.
- $\text{TIME}(f(n))$ is the set of languages decided by TMs with multiple strings operating within time bound $f(n)$.
- $\text{TIME}(f(n))$ is a **complexity class**.
  - PALINDROME is in $\text{TIME}(f(n))$, where $f(n) = O(n)$.

[a]Hartmanis and Stearns (1965), Hartmanis, Lewis, and Stearns (1965).

The Simulation Technique

Theorem 2 *Given any k-string M operating within time $f(n)$, there exists a (single-string) M' operating within time $O(f(n)^2)$ such that $M(x) = M'(x)$ for any input x.*

- The single string of M' implements the k strings of M.
- Represent configuration $(q, w_1, u_1, w_2, u_2, \ldots, w_k, u_k)$ of M by configuration

$$(q, \triangleright w_1' u_1 \triangleleft w_2' u_2 \triangleleft \cdots \triangleleft w_k' u_k \triangleleft \triangleleft)$$

  of M'.
  - $\triangleleft$ is a special delimiter.
  - w_i' is w_i with the first and last symbols "primed."

The Proof (continued)

- The initial configuration of M' is

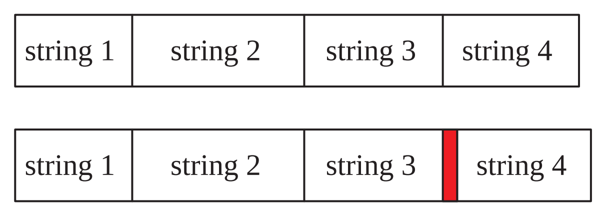
$$(s, \triangleright \triangleright' x \triangleleft \overbrace{\triangleright' \triangleleft \cdots \triangleright' \triangleleft}^{k-1 \text{ pairs}} \triangleleft).$$

- To simulate each move of M:
  - M' scans the string to pick up the k symbols under the cursors.
    * The states of M' must include $K \times \Sigma^k$ to remember them.
    * The transition functions of M' must also reflect it.
  - M' then changes the string to reflect the overwriting of symbols and cursor movements of M.

The Proof (continued)

- It is possible that some strings of M need to be lengthened.
  - The linear-time algorithm on p. 22 can be used for each such string.
- The simulation continues until M halts.
- M' erases all strings of M except the last one.
- Since M halts within time $f(|x|)$, none of its strings ever becomes longer than $f(|x|)$.[a]
- The length of the string of M' at any time is $O(kf(|x|))$.

[a] We tacitly assume $f(n) \geq n$.



The Proof (concluded)

- Simulating each step of M takes, *per string of* M, $O(kf(|x|))$ steps.
  - $O(f(|x|))$ steps to collect information.
  - $O(kf(|x|))$ steps to write and, if needed, to lengthen the string.
- M' takes $O(k^2 f(|x|))$ steps to simulate each step of M.
- As there are $f(|x|)$ steps of M to simulate, M' operates within time $O(k^2 f(|x|)^2)$.

Linear Speedup[a]

Theorem 3 *Let* $L \in TIME(f(n))$*. Then for any* $\epsilon > 0$*,* $L \in TIME(f'(n))$*, where* $f'(n) = \epsilon f(n) + n + 2$*.*

- If $f(n) = cn$ with $c > 1$, then c can be made arbitrarily close to 1.
- If $f(n)$ is superlinear, say $f(n) = 14n^2 + 31n$, then the constant in the leading term (14 in this example) can be made arbitrarily small.
  - *Arbitrary* linear speedup can be achieved.
  - This justifies the asymptotic big-O notation.

[a]Hartmanis and Stearns (1965).