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.
  - We more or less follow the topics of the book.
  - Extra materials may be added.
- You may want to review discrete mathematics.[a]

[a] `www.csie.ntu.edu.tw/~lyuu/dm.html`

Class Information (continued)

- More information and lecture notes can be found at

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

  - (Homeworks,) exams, solutions and teaching assistants will be announced there.
  - Past homeworks and solutions to past exams can be found there, too.

Class Information (concluded)

- Please ask many questions in class.
  - This is the best way for me to remember you in a large class.[a]

[a] Lawrence H. Summers, "[A] science concentrator [...] said that in his eighth semester of [Harvard] college, there was not a single science professor who could identify him by name." (*New York Times*, September 3, 2003.)

Grading

- Three exams.
- You must show up for the exams in person.
- If you cannot make it to an exam for a legitimate reason, please email me or a TA beforehand to the extent possible.

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

Computation: What is computation?

Computability: What can be computed?

- There are problems that cannot be computed.
- In fact, most problems cannot be computed.

What This Course Is All About (continued)

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

- Some computable problems require at least exponential time and/or space.
  - They are said to be **intractable**.
- Some practical problems require superpolynomial[a] resources unless certain conjectures are disproved.
- Resources besides time and space: Circuit size, circuit layout area, program size, number of random bits, number of queries, etc.

[a]The prefix "super" means "above, beyond."

What This Course Is All About (concluded)

Applications: Intractability results can be very useful.

- Cryptography, digital currency, and security.
- Approximations.
- Pseudorandom number generation and derandomization.
- Even conjectures about nature.

Tractability and Intractability

- Tractability means polynomial in terms of the input size n.
  - n, $n \log n$, n^2, n^{90}.
- It results in a fruitful and practical theory of complexity.
- Few practical, tractable problems require a large degree.
- Superpolynomial-time algorithms are seldom practical.
  - $n^{\log n}$, $2^{\sqrt{n}}$,[a] 2^n, $n! \sim \sqrt{2\pi n}\,(n/e)^n$.

[a] Size of depth-3 circuits to compute the majority function (Wolfovitz, 2006) and certain stochastic models used in finance (Dai (R86526008, D8852600) & Lyuu, 2007; Lyuu & Wang (F95922018), 2011; Chiu (R98723059), 2012).

Exponential Growth of *E. Coli*[a]

- Under ideal conditions, *E. Coli* bacteria divide every 20 minutes.
- In two days, a single *E. Coli* bacterium would become 2^{144} bacteria.
- They would weigh 2,664 times the Earth!

[a] Nick Lane (2005), *Power, Sex, Suicide: Mitochondria and the Meaning of Life*.

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

Moore's Law to the Rescue?[a]

- One version of Moore's law says the computing power doubles every 1.5 years.[b]
- So the computing power grows like

$$4^{y/3},$$

  where y is the number of years from now.
- Assume Moore's law holds forever.
- Can we let the law tame exponential complexity?

---

[a] Contributed by Ms. Amy Liu (J94922016) on May 15, 2006. Thanks also to a lively discussion on September 14, 2010.

[b] Moore (1965). Bitcoin implicitly assumes computing power doubles every 4 years (Nakamoto, 2009).

Moore's Law to the Rescue (continued)?

- Suppose a problem takes a^n seconds of CPU time to solve now, where n is the input length and $a > 1$.
- The same problem will take

$$\frac{a^n}{4^{y/3}}$$

  seconds to solve y years from now.
- In particular, the hardware $3n \log_4 a$ years from now takes 1 second to solve it.
- The overall complexity becomes linear in n![a]

[a] $3n \log_4 a$ years plus 1 second.

Moore's Law to the Rescue (concluded)?

- Potential objections:
  - Moore's law may not hold forever.
  - The total number of operations is the same; so the *algorithm* remains exponential in complexity.[a]
- What is a "good" theory on computational complexity?
  - Should it be based on technology?
  - Or should it be based on mathematics?

[a]Contributed by Mr. Hung-Jr Shiu (D00921020) on September 14, 2011.

Turing Machines

Tarski has stressed in his lecture
(and I think justly)
the great importance of
the concept of general recursiveness
(or Turing's computability).
— Kurt Gödel (1946)

Either mathematics is too big
for the human mind, or the human mind
is more than a machine.
— Kurt Gödel[a]

[a]Goldblatt (1979).

What Is Computation?

- That can be coded in an **algorithm**.[a]
- An algorithm is a detailed step-by-step method for solving a problem.
  - The Euclidean algorithm for the greatest common divisor is an algorithm.
  - Addition, multiplication, and division can be solved by algorithms.
  - How about passing the Turing test?

[a] Muhammad ibn Mūsā Al-Khwārizmī (780–850).

Turing Machines[a]

- A Turing machine (TM) is a quadruple $M = (K, \Sigma, \delta, s)$.
- K is a finite set of **states**.[b]
- $s \in K$ is the **initial state**.
- Σ is a finite set of **symbols** (disjoint from K).
  - Σ includes $\sqcup$ (blank) and $\triangleright$ (first symbol).[c]

---

[a] Turing (1936, 1937); Post (1936).

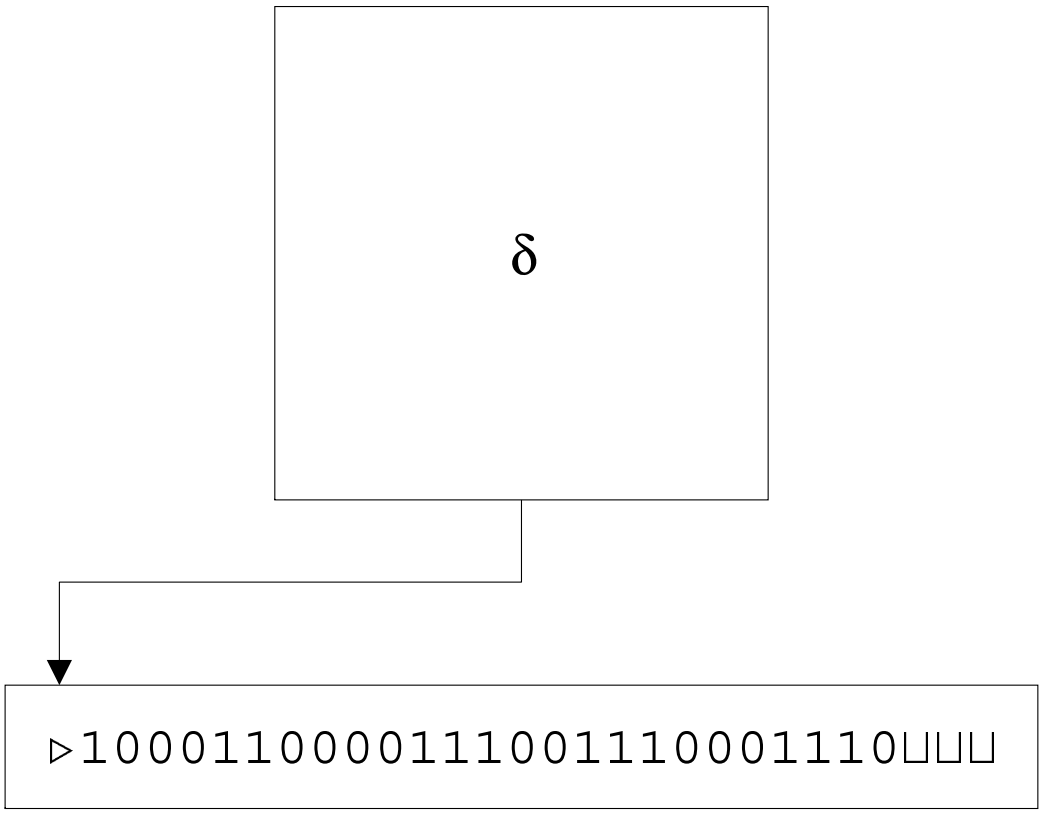
[b] Turing (1936), "If we admitted an infinity of states of mind, some of them will be 'arbitrarily close' and will be confused." In any case, every physical device (lens, microscope, sensor, etc.) has limited resolving power. Thanks to a lively discussion on February 21, 2019.

[c] Two special symbols as we will see.

Turing Machines (concluded)

- $\delta : K \times \Sigma \to (K \cup \{h, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$ is a **transition function**.
  - $\leftarrow$ (left), $\rightarrow$ (right), and $-$ (stay) signify cursor movements.

A TM Schema



An Open-Reel Recorder

More on 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;
    - The direction D the cursor will move *afterwards*.

More on δ (continued)

- For convenience, $\delta(q, \triangleright) = (\cdot, \triangleright, \rightarrow)$ for every $q \in K$.
  - So the cursor never falls off the left end of the string.
- Think of the program as (a soup of) lines of codes:

$$\begin{aligned} \delta(q_1, \sigma_1) &= (p_1, \rho_1, D_1), \\ \delta(q_2, \sigma_2) &= (p_2, \rho_2, D_2), \\ &\vdots \\ \delta(q_n, \sigma_n) &= (p_n, \rho_n, D_n). \end{aligned}$$

- Their order is unimportant.

More on δ (concluded)

- Assume the state is q and the symbol under the cursor σ.
- The line of code that matches (q, σ) is fired/triggered/executed.[a]
- Then the process is repeated.

[a] So there should be at most one instruction for every possible pair (q, σ). Contributed by Mr. Ya-Hsun Chang (B96902025, R00922044) on September 13, 2011.

The Operations of TMs

- Initially the state is s.
- The string on the tape is initialized to a $\triangleright$, followed by a *finite-length* string $x \in (\Sigma - \{\sqcup\})^*$.[a]
- 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 $\triangleright$.
- The TM takes each step according to δ.
- The cursor may overwrite $\sqcup$ to lengthen the string.
- Writing down a $\sqcup$ amounts to erasure.

[a] See p. 50 for the definition of $*$.

"Physical" Correspondences

- The tape: computer memory and registers.
  - Except that the tape can be lengthened on demand.
- δ: program.
  - A program has a *finite* size.
- K: instruction numbers.
- s: "`main()`" in the C programming language.
- Σ: **alphabet**, much like the ASCII code.

Alan Turing (1912–1954)

Richard Dawkins (2006), "Turing arguably made a greater contribution to defeating the Nazis than Eisenhower or Churchill."

Michael Peck (2014), "But ULTRA didn't detect German preparations, which was taken as an indication that nothing was happening."

The Halting of a TM

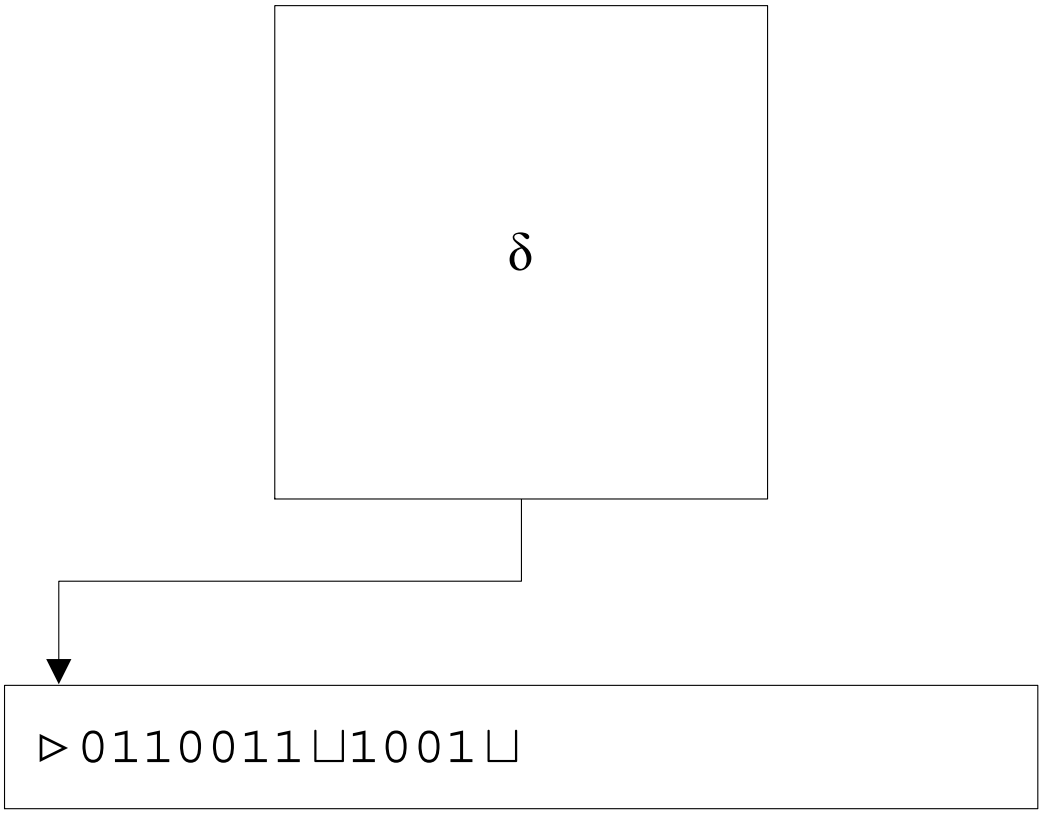
- A TM M may **halt** in three cases.

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

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

  h**:** $M(x) = y$ means the string (tape) consists of a $\triangleright$, followed by the finite string y which contains no $\sqcup$s, followed by a $\sqcup$.
  - y is the **output** of the computation.
  - y may be empty denoted by ϵ.

- If M never halts on x, then write $M(x) = \nearrow$.



- $y = 0110011$.

The First TM Program[a]

- Assume $M = (K, \Sigma, \delta, s)$, where $K = \{ s, h \}$, $\Sigma = \{ 0, 1, \sqcup, \triangleright \}$, and

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
s	1	$(s, 0, \rightarrow)$
s	0	$(s, 1, \rightarrow)$
s	$\sqcup$	$(h, \sqcup, -)$

- This TM converts all 1's in the input string to 0's and vice versa.

[a]Contributed by Mr. Zheyuan (Jeffrey) Gao (`R01922142`) on September 21, 2013.

The Second TM Program[a]

- Assume $M = (K, \Sigma, \delta, s)$, where $K = \{ s, s_1, h \}$, $\Sigma = \{ 0, 1, \sqcup, \triangleright \}$, and

[a] Contributed by Mr. Zheyuan (Jeffrey) Gao (R01922142) on September 21, 2013.

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
s	0	$(s, 0, \rightarrow)$
s	1	$(s_1, 1, \rightarrow)$
s_1	0	$(s, 0, \rightarrow)$
s_1	1	$(h, 1, -)$
s	$\sqcup$	$(h, \sqcup, -)$
s_1	$\sqcup$	$(h, \sqcup, -)$

The Second TM Program (concluded)

- This TM scans to the right until it finds two consecutive 1's and then halts.
- Otherwise, it halts at the end of the input string.
- But it does not tell you its findings.

The Third TM Program

- Assume $M = (K, \Sigma, \delta, s)$, where
  $K = \{ s, s_1, \text{“yes”}, \text{“no”} \}$, $\Sigma = \{ 0, 1, \sqcup, \triangleright \}$, and

$p \in K$	$\sigma \in \Sigma$	$\delta(p, \sigma)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
s	0	$(s, 0, \rightarrow)$
s	1	$(s_1, 1, \rightarrow)$
s_1	0	$(s, 0, \rightarrow)$
s_1	1	$(\text{“yes”}, 1, -)$
s	$\sqcup$	$(\text{“no”}, \sqcup, -)$
s_1	$\sqcup$	$(\text{“no”}, \sqcup, -)$

The Third TM Program (concluded)

- This TM accepts the input if there are two consecutive 1's.
- Otherwise, it rejects the input string.

Why Turing Machines?

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

[a] But you are strongly encouraged to read and understand the TM codes in the textbook to gain insight on its subtleties.

A TM Program To Insert a Symbol

- We want to compute $f(x) = ax$.
  - The TM moves its cursor to the last symbol.
  - It moves the last symbol of x to the right by one position.
  - It 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.

Remarks

- A computation model should be "physically" realizable.
  - E.g., our brain, at least as powerful as a Turing machine, is physical.
- A TM requires a tape of unbounded length, which is not realizable.
- But it is not a major *conceptual* issue.[a]
  - Imagine you ("the program") live next to a paper mill while carrying out a TM code using pencil ("the cursor") and paper ("the tape").
  - The mill will produce extra paper if needed.

[a] Thanks to a lively discussion on September 20, 2006.

Remarks (concluded)

- Even our computer is only an approximation of a TM.
- But it is easy to imagine our computer with more and more address space, memory space, and disk space.

The Concept of Configuration

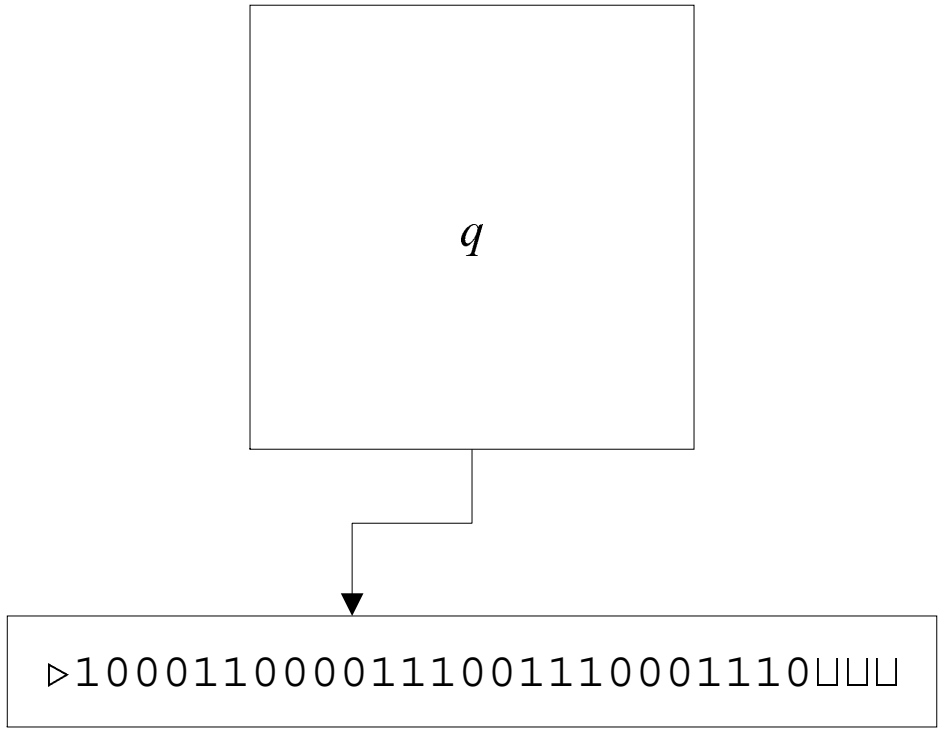
- A **configuration**[a] is a complete description of the current state of the computation.
- The specification of a configuration is sufficient for the computation to continue *as if it had not been stopped*.
  - What does your PC save before it sleeps or hibernates?
  - Enough for it to resume the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

[a] This term was due to Turing (1936).

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.
    - Again, u has a finite length because the trailing $\sqcup$s are not needed.[a]
- Note that (w, u) describes both the string and the cursor position (implicitly).

[a] Thanks to a lively discussion on September 23, 2021.



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

Yielding

- Fix a TM M.
- Configuration (q, w, u) **yields** configuration (q', w', u') in one step,

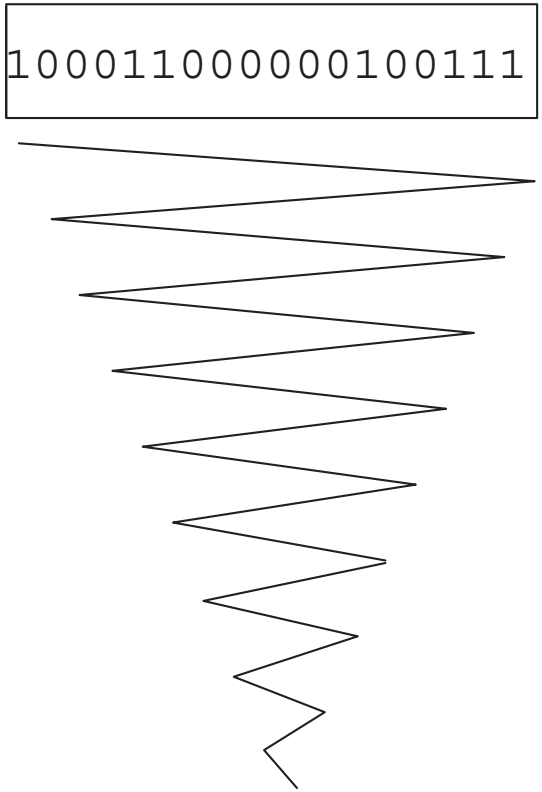
  $$(q, w, u) \xrightarrow{M} (q', w', u'),$$

  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') after $k \in \mathbb{N}$ steps.
- $(q, w, u) \xrightarrow{M^*} (q', w', u')$: Configuration (q, w, u) yields configuration (q', w', u').

Palindromes[a]

- 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?

[a]Bryson (2001), "Possibly the most demanding form of wordplay in English[.]"


1000110000000100111

A Matching Lower Bound for PALINDROME

Theorem 1 (Hennie, 1965) PALINDROME *on single-string TMs takes* $\Omega(n^2)$ *steps in the worst case.*

Comments on Lower-Bound Proofs

- They are usually difficult.
  - Worthy of a Ph.D. degree.
- An algorithm whose running time matches a lower bound means it is optimal.
  - The simple $O(n^2)$ algorithm for PALINDROME is optimal.
- This happens rarely and is model dependent.
  - Searching, sorting, PALINDROME, matrix-vector multiplication, etc.

The Kleene Star[a] $*$

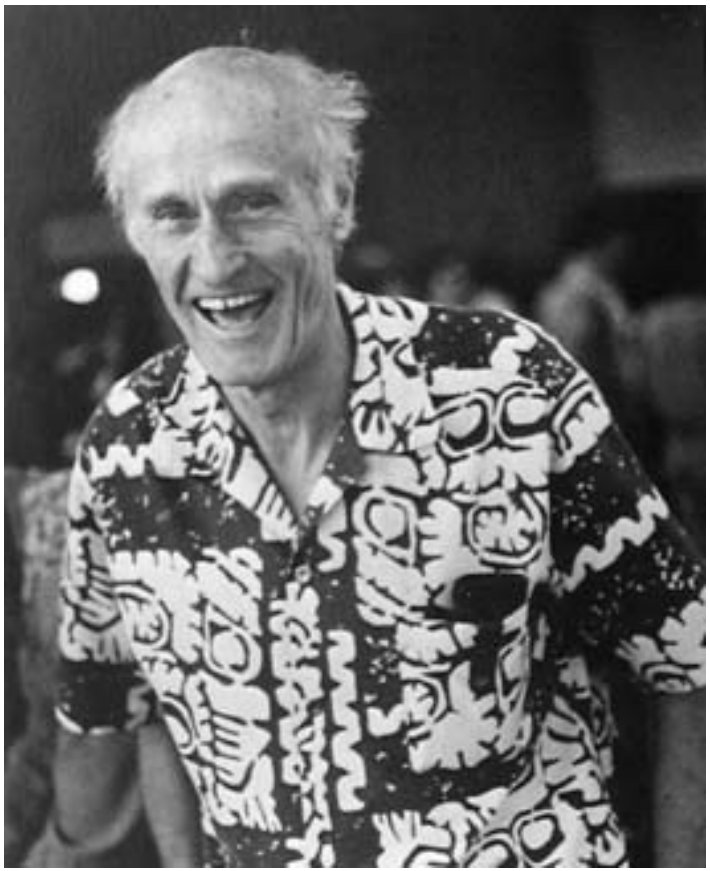
- Let A be a set.
- The **Kleene star** of A, denoted by A^*, is the set of all strings obtained by concatenating zero or more strings from A.
  - For example, suppose $A = \{ 0, 1 \}$.
  - Then

$$A^* = \{ \epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots \}.$$

  - Note that every string in A^* is of finite length.

[a] Kleene (1956).

Stephen Kleene (1909–1994)



The two words in the language I most respect
are Yes and No.
— Henry James (1843–1916),
The Portrait of a Lady (1881)

Decidability and Recursive Languages

- Let $L \subseteq (\Sigma - \{\sqcup\})^*$ be a **language**, i.e., a set of strings of non-$\sqcup$ symbols, with a *finite* length.
  - For example, $\{ 2, 3, 5, 7, 11, \ldots \}$ (the primes).
- 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 there exists a TM that decides L, then L is said to be **recursive**[a] or **decidable**.

[a] Little to do with the concept of "recursive" calls.

Recursive and Nonrecursive Languages: Examples

- The set of palindromes over any alphabet is recursive.[a]
  - PALINDROME cannot be solved by finite state automata.
  - In fact, finite-state automata are equivalent to read-only, right-moving TMs.[b]
- The set of prime numbers $\{ 2, 3, 5, 7, 11, 13, 17, \ldots \}$ is recursive.[c]

---

[a] There is a program that will halt and it returns "yes" if and only if the input is a palindrome.

[b] Thanks to a lively discussion on September 15, 2015.

[c] There is a program that will halt and it returns "yes" if the input is a prime and "no" othrerwise.

Recursive and Nonrecursive Languages: Examples (concluded)

- The set of C programs that do not contain a `while`, a `for`, or a `goto` is recursive.[a]
- But, the set of C programs that do not contain an infinite loop is *not* recursive.[b]

[a] There is a program that will halt and it returns "yes" if and only if the input C code does not contain any of the keywords.

[b] So there is no algorithm that will answer correctly in a finite amount of time if a C program will run into an infinite loop on some inputs (see p. 145).

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 $M(x) = \nearrow$.[a]
- We say M **accepts** L.
- If L is accepted by some TM, then L is said to be **recursively enumerable** or **semidecidable**.[b]

[a] This part differs from recursive languages.

[b] Post (1944).

Acceptability and Recursively Enumerable Languages (concluded)

- A recursively enumerable language can be *generated* by a TM, thus the name.[a]
  - It means there is a program such that every $x \in L$ (and only they) will be printed out eventually.
- Of course, if L is infinite in size, this program will not terminate.

[a] Proposition 3.5 on p. 61 of the textbook proves it. Thanks to lively class discussions on September 20, 2011, and September 12, 2017.

Emil Post (1897–1954)

W. V. Quine (1985), "E. L. Post worked alone in New York, little heeded."

Recursive and Recursively Enumerable Languages

Proposition 2 *If L is recursive, then it is recursively enumerable.*

- Let TM M decide L.
- Need to design a TM that accepts L.
- We will modify M to obtain an M' that accepts L.

The Proof (concluded)

- M' is identical to M except that when M is about to halt with a "no" state, M' goes into an infinite loop.
  - Simply replace every instruction that results in a "no" state with ones that move the cursor to the right forever and never halts.
- M' accepts L.
  - If $x \in L$, then $M'(x) = M(x) =$ "yes."
  - If $x \notin L$, then $M(x) =$ "no" and so $M'(x) = \nearrow$.

Recursively Enumerable Languages: Examples

- The set of C program-input pairs that do *not* run into an infinite loop is recursively enumerable.
  - Just run its binary code in a simulator environment.
  - Then the simulator will terminate if and only if the C program will terminate.
  - When the C program terminates, the simulator simply exits with a "yes" state.
- The set of C programs that can run into an infinite loop is *not* recursively enumerable.[a]

[a] See p. 165.

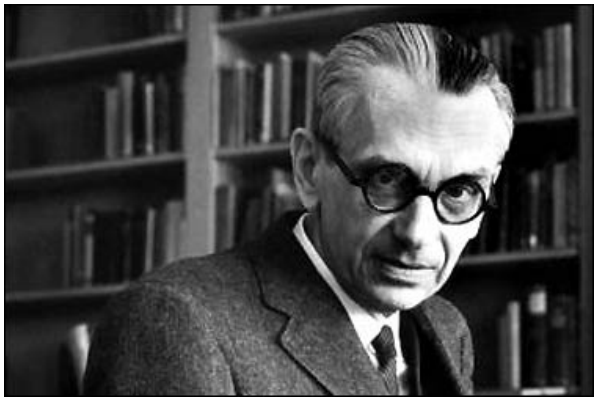
Turing-Computable Functions

- 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 $x \in (\Sigma - \{\sqcup\})^*$, $M(x) = f(x)$.
  - f may be a *partial* function.
  - Then $f(x)$ is undefined if $M(x)$ diverges.
- We call f a **(partial) recursive function**[a] if such an M exists.

[a] Gödel (1931, 1934); Kleene (1936).

Kurt Gödel[a] (1906–1978)

Quine (1978), "this theorem [$\cdots$] sealed his immortality."



[a]This photo was taken by Alfred Eisenstaedt (1898–1995).

Church's Thesis

- What is computable is Turing-computable; TMs are algorithms.[a]
- No "intuitively computable" problems have been shown not to be Turing-computable (yet).[b]

[a] Church (1936); Kleene (1943, 1953).

[b] Quantum computer of Manin (1980) and Feynman (1982); DNA computer of Adleman (1994).

Church's Thesis (continued)

- Many other computation models have been proposed.
  - **Recursive function**,[a] λ calculus,[b] **boolean circuits**,[c] **formal language**,[d] **assembly language-like RAM**,[e] **cellular automaton**,[f] **recurrent neural network**,[g] and extensions of the Turing machine (more strings, two-dimensional strings, etc.).

---

[a] Skolem (1923); Gödel (1934); Kleene (1936).
[b] Church (1936).
[c] Shannon (1937).
[d] Post (1943).
[e] Shepherdson & Sturgis (1963).
[f] Conway (1970).
[g] Siegelmann & Sontag (1991).

Church's Thesis (concluded)

- All have been proved to be equivalent.
- Church's thesis is also called the **Church-Turing Thesis**.

Alonso Church (1903–1995)

Extended Church's Thesis[a]

- All "reasonably succinct encodings" of problems are *polynomially related* (e.g., n^2 vs. n^6).
  - 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_2 vs. 111111111_1.
- All numbers for TMs will be binary from now on.

[a]Some call it "polynomial Church's thesis," which Lószló Lovász attributed to Leonid Levin (1948–).

Extended Church's Thesis (concluded)

- Representations that are not succinct may give misleadingly low complexities.
  - Consider an algorithm with binary inputs that runs in 2^n steps.
  - Suppose the input uses unary representation instead.
  - Then the same algorithm runs in *linear* time because the input length is now 2^n!
- So a succinct representation means honest accounting.

Physical Church-Turing Thesis

- The **physical Church-Turing thesis** states that:

  Anything computable in physics can also be computed on a Turing machine.[a]

- The universe is a Turing machine.[b]

[a] Cooper (2012).

[b] Edward Fredkin's (1992) digital physics.

The Strong Church-Turing Thesis[a]

- The **strong Church-Turing thesis** states that:[b]

  A Turing machine can compute *any* function computable by any "reasonable" physical device with only polynomial slowdown.[c]

- A CPU, a GPU, and a DSP chip are good examples of physical devices.[d]

---

[a] Vergis, Steiglitz, & Dickinson (1986).

[b] `http://ocw.mit.edu/courses/mathematics/18-405j-advanced-complexity-theory-fall-2001/lecture-notes/lecture10.pdf`

[c] Or speedup.

[d] Thanks to a lively discussion on September 23, 2014.

The Strong Church-Turing Thesis (continued)

- Factoring is believed to be a hard problem for Turing machines (but there is no proof yet).
- But a quantum computer can factor numbers in probabilistic polynomial time.[a]
- If a large-scale stable quantum computer can be reliably built, the strong Church-Turing thesis may be refuted.[b]

---

[a] Shor (1994).

[b] Contributed by Mr. Kai-Yuan Hou (`B99201038`, `R03922014`) on September 22, 2015.

The Strong Church-Turing Thesis (concluded)

- As of 2019,[a]

  There is no publicly known application of commercial interest based upon quantum algorithms that could be run on a near-term analog or digital NISQ[b] computer that would provide an advantage over classical approaches.

[a] Grumbling & Horowitz (2019).

[b] "Noisy, Intermediate-Scale Quantum."