Theory of Computation Class Notes

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

Class Information

- *Computational Complexity*, 2nd printing, 1995, by Papadimitriou.
  - Arguably the best book on the market for graduate students.
- Probably no homework.
- At least two examinations.
- No roll calls.
  - You do not have to show up for class.
  - You do have to show up for examinations, in person.
- Teaching assistants to be announced.

A Brief History (Biased towards Complexity)

1930–1931: Gödel's (1906–1978) completeness and incompleteness theorems.

1935–1936: Kleene (1909–1994), Turing (1912–1954), Church (1903–1995), Post (1997–1954) on computability.

1936: Turing defined Turing machines and oracle Turing machines.

1938: Shannon (1916–2001) used boolean algebra for the design and analysis of switching circuits. Circuit complexity was also born. Shannon's master's thesis was "possibly the most important, and also the most famous, master's thesis of the century."

A Brief History (continued)

1947: Dantzig invented linear programming simplex algorithm.

1947: Paul Erdős (1913–1996) popularized the probabilistic method. (Also Shannon, 1948.)

1949: Shannon established information theory.

1949: Shannon's study of cryptography was published.

1956: Ford and Fulkerson's network flows.

1959: Rabin and Scott's notion of nondeterminism.

A Brief History (Biased towards Complexity)

1964–1966: Solomonoff, Kolmogorov, and Chaitin formalized Kolmogorov complexity (program size and randomness).

1965: Hartmanis and Stearns started complexity theory and hierarchy theorems. (See also Rabin, 1960.)

1965: Edmonds identified NP and P (actual names were coined by Karp in 1972).

1971: Cook invented the idea of NP-completeness.

1972: Karp established the importance of NP-completeness.

1972–1973: Karp, Meyer, and Stockmeyer defined the polynomial hierarchy.

A Brief History (Biased towards Complexity)

1973: Karp studied PSPACE-completeness.

1973: Meyer and Stockmeyer studied exponential time and space.

1973: Baker, Gill, and Solovay studied "NP=P" relative to oracles.

1975: Ladner studied P-completeness.

1976–1977: Rabin, Solovay, Strassen, and Miller proposed probabilistic algorithms (for primality testing).

1976–1978: Diffie, Hellman, and Merkle invented public-key cryptography.

A Brief History (Biased towards Complexity)

1977: Gill formalized randomized complexity classes.

1978: Rivest, Shamir, and Adleman invented RSA.

1978: Fortune and Wyllie defined the PRAM model.

1979: Garey and Johnson published their book on computational complexity.

1979: Valiant defined #P.

1979: Pippenger defined NC.

1979: Khachiyan proved that linear programming is in polynomial time.

1979: Yao founded communication complexity.

A Brief History (Biased towards Complexity)

1980: Lamport, Shostak, and Pease defined the Byzantine agreements problem in distributed computing.

1981: Shamir proposed cryptographically strong pseudorandom numbers.

1982: Goldwasser and Micali proposed probabilistic encryption.

1982: Yao founded secure multiparty computation.

1982: Goldschlager, Shaw, and Staples proved that the maximum flow problem is P-complete.

1982–1984: Yao, Blum, and Micali founded pseudorandom number generation on complexity theory.

A Brief History (Biased towards Complexity)

1983: Ajtai, Komlós, and Szemerédi constructed an $O(\log n)$-depth, $O(n \log n)$-size sorting network.

1984: Valiant founded computational learning theory.

1984–1985: Furst, Saxe, Sipser, and Yao proved exponential bounds for parity circuits of constant depth.

1985: Razborov proved exponential lower bounds for monotone circuits.

1985: Goldwasser, Micali, and Rackoff invented zero-knowledge proofs.

1985: Sleator and Tarjan invented on-line algorithms.

A Brief History (Biased towards Complexity)

1987–1988: Szelepscényi and Immerman proved that NL equals coNL.

1989: Blum and Kannan proposed program checking.

1990: Shamir proved IP=PSPACE.

1990: Du and Hwang settled the Gilbert-Pollak conjecture on Steiner tree problems.

1992: Arora, Lund, Motwani, Sudan, and Szegedy proved the PCP theorem.

What This Course Is All About

Computability: What can be computed?

- There exist well-defined problems that cannot be computed.
- In fact, "most" problems cannot be computed.

Complexity: What is the inherent complexity of problems?

- Some computable problems require exponential time and/or space; they are **intractable**.
- Some practical problems require super-polynomial resources unless certain conjectures are disproved.
- What if we impose more limits besides time and space?

Tractability and intractability

- Polynomial in terms of the input size n defines tractability.
  - n, $n \log n$, n^2, n^{90}.
- Time (more often), space, circuit size, etc.
- It results in a fruitful and practical theory of complexity.
- Few practical, tractable problems require a large degree.
- Exponential-time algorithms are usually impractical unless we compromise on complete "correctness."
  - $n^{\log n}$, $2^{\sqrt{n}}$, 2^n, $n!$.

Most Important Results: A Sampler

- An operational definition of computability.
- Decision problems in logic are undecidable.
- Decisions problems on program behavior are usually undecidable.
- Complexity classes and the existence of intractable problems.
- Identification of complete problems for a complexity class.
- Randomization and cryptographic applications.
- Nonapproximability.

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.

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 $\sqcup$ (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.

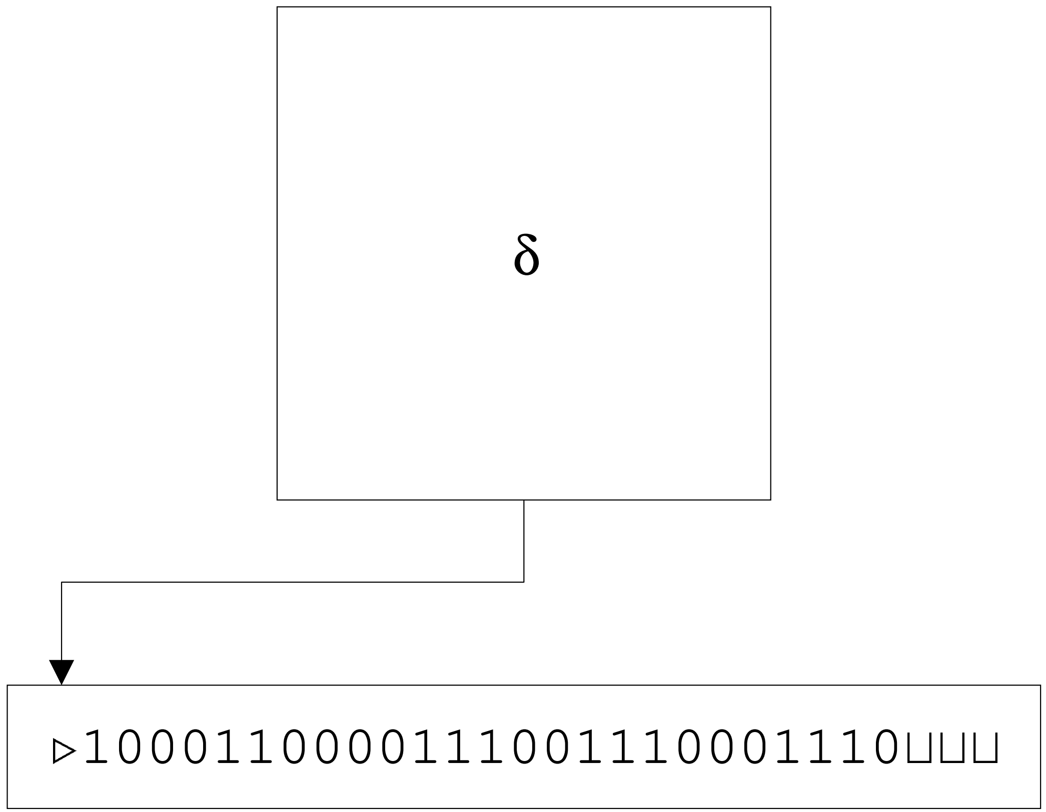
“Physical” Interpretations

- K is like instruction numbers.
- s is like “main()” in C.
- Σ is the **alphabet**.
- δ is the program with the halting state (h), the accepting state (“yes”), and the rejecting state (“no”).
  - Given the current state $q \in K$ and the current symbol $\sigma \in \Sigma$,
  $$\delta(q, \sigma) = (p, \rho, D)$$
  specifies the next state p, the symbol ρ to be written over σ, and the direction D the cursor will move afterwards.
  - We require $\delta(q, \rhd) = (p, \rhd, \rightarrow)$.

The Operations of TMs

- Initially the state is s.
- The string on the tape is initialized to a $\triangleright$, followed by a finitely long string $x \in (\Sigma - \{\sqcup\})^*$.
- x is the **input** of the TM.
- The cursor is pointing to the first symbol, always a $\triangleright$.
- The TM takes each step according to δ.
- The cursor never falls off the left end of the string.
- The cursor may overwrite $\sqcup$ to make the string longer during the computation.

A TM Schematic



Program Size

- Recall that the program δ is a function from $K \times \Sigma$ to $(K \cup \{h, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}$.
- To completely specify such a function, $|K| \times |\Sigma|$ states suffice.
- Given K and Σ, there are

$$((|K| + 3) \times |\Sigma| \times 3)^{|K| \times |\Sigma|}$$

  possible δ's, a constant—albeit large.
  - All programs have a finite size.
- Different δ's may define the same TM.

The Halting of a TM

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

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

  "no": The machine **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, if any, by a string of $\sqcup$s (y may be empty denoted by ϵ).

  – y is called the **output** of the computation.

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