Foundations of Computability Theory (1)

Nan Fang

Institute of Software, Chinese Academy of Sciences

Peking University

2025 Autumn Semester

What is computability?

- An algorithm effectively maps an instance x from a problem set P to its solution y.
- For problems coded with natural numbers, such map always exists. The question is: can it be computable?

Then what is a computable function?

In the 1930s, several approaches were developed to formalize the concept of computability:

- General recursive functions
- Lambda calculus
- Turing machines

Remarkably, all these formalizations were shown to define the same class of functions. This led mathematicians to believe that they had successfully captured the essence of computability, a principle now known as the **Church-Turing Thesis**.

Overview of computability theory

- The main object in computability are the non-computable functions, as computable function are kind of trivial in the sense of computability.
- Non-computable functions can be classified to different (Turing) degrees according to their computational power. The structure of Turing degrees is the main theme in the study of classical computability theory

Current research areas of computability theory include:

- Classical computability theory
- Algorithmic randomness
- Reverse mathematics
- Computable analysis
- Higher computability theory
- Computable structure theory
- Degrees of enumeration
- ...

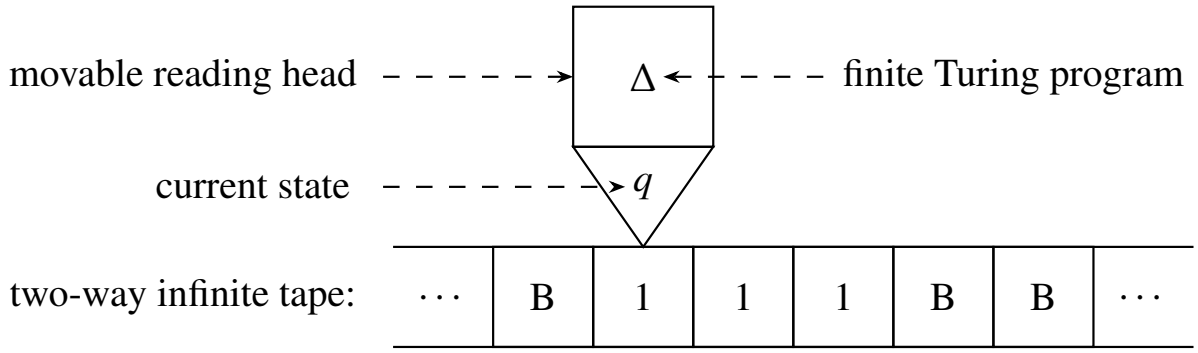
The area of computational complexity theory is closely related to computability theory in the beginning, but has diverged a lot in the past decades. The methods and techniques are quite different.

Plan of this lecture

We will try to cover the following basic topics:

1. Turing machines, s-m-n theorem, recursion theorem, c.e. sets, halting problem, m-reducibility and 1-reducibility, degree structure, index sets, Rices Theorem, hardness and completeness [W1-3]
2. oracle Turing machine, Turing reducibility, use principle, relativization, Turing degrees, Turing jump, Jump Theorem, bT-reducibilities, tt-reducibilities, c.a. sets, omega-c.e. and n-c.e. sets, limit lemma [W4-5]
3. arithmetical hierarchy, Post's Theorem, low and high sets, dominating sets, Π^0_1 classes, basis theorems, PA degrees [W6-7]
4. simple sets, immunity, DNC functions, dominating and escaping functions [W8-9]
5. generic sets, effective forcing, finite extension method [W10-11]
6. Post's problem, priority method [W12-13]
7. Gödel incompleteness theorem [W14-15]

Turing machines



Definition 1.1

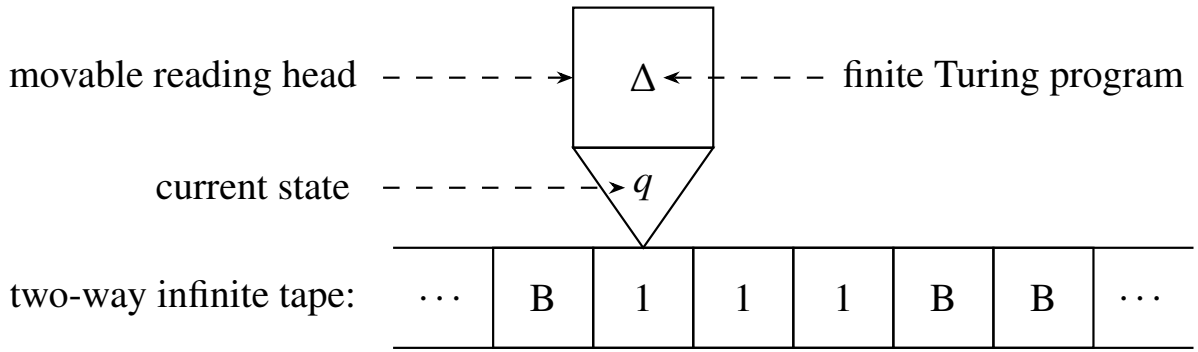
A *Turing machine M* consists of

- a two-way infinite tape divided into cells, with each cell marked as B or 1,
- a reading head which scans one cell of the tape at a time,
- a finite set of internal states $Q = \{q_0, q_1, \ldots, q_n\}, n \geq 1$,
- a finite set of instructions $\Delta = \{\delta_0, \delta_1, \ldots, \delta_m\}, m \geq 0$.

Each instruction $\delta \in \Delta$ is a 5-tuple of the form $\delta = (q_i, s, q_j, s', D)$, where $q_i, q_j \in Q$ are states, $s, s' \in S = \{B, 1\}$ are symbols, and $D \in \{L, R\}$ is a direction (left or right).

Δ can be seen as a partial map from $Q \times S$ to $Q \times S \times \{L, R\}$.

Turing machines



- An instruction $\delta = (q_i, s, q_j, s', D)$ means that if the machine is in state q_i and the reading head scans a cell containing symbol s, then the machine will change to state q_j, write symbol s' in the scanned cell, and move the reading head one cell in direction D.
- The input natural number x is represented by a string of $x + 1$ consecutive 1s (with all other cells blank).
- M starts with *starting state* q_1 scanning the leftmost cell containing a 1. If the machine ever reaches the *halting state* q_0, after say s steps, then we say M halts and the output y is the total number of 1s on the tape.
- We say that M *computes* the partial function ψ provided that $\psi(x) = y$ if and only if M with input x eventually halts and yields output y.

Turing machines: example

Fact 1.2

A Turing machine is determined by its instruction set.

For example, the following machine computes the function $f(x) = x + 3$.

$$(q_1, 1, q_1, 1, R)$$
$$(q_1, B, q_2, 1, R)$$
$$(q_2, B, q_0, 1, R)$$

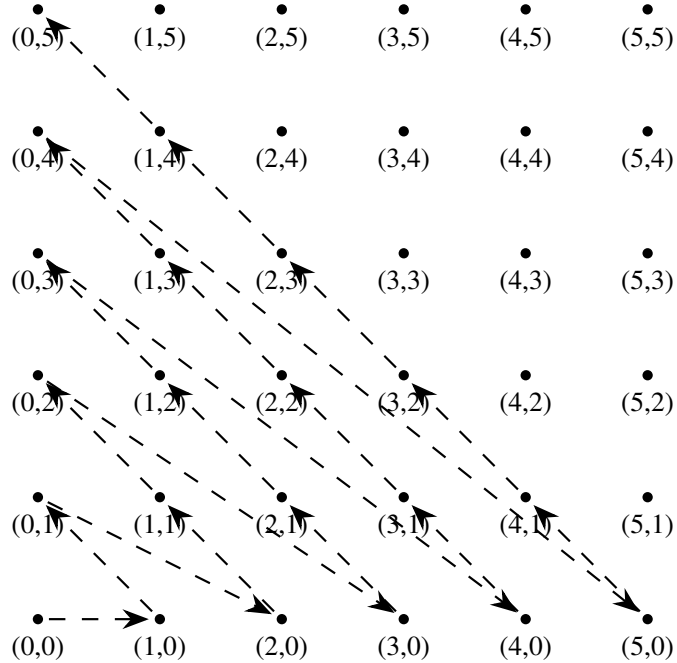
Definition 1.3

A partial function $f : \mathbb{N} \to \mathbb{N}$ is *(Turing) computable* if there exists a Turing machine computes f.

Numbering of Finite Objects

Definition 1.4 (Pairing Function)

1. $\langle x, y\rangle := y + \sum_{i=0}^{x+1} i = \frac{1}{2}(x^2 + 2xy + y^2 + x + 3y)$. Then $(x, y) \mapsto \langle x, y\rangle$ is a 1:1 computable function from $\mathbb{N} \times \mathbb{N}$ onto $\mathbb{N}$. Let π_1 and π_2 denote the *inverse pairing functions* $\pi_0(\langle x, y\rangle) = x$, and $\pi_1(\langle x, y\rangle) = y$.
2. $\langle x_1, x_2, x_3\rangle := \langle\langle x_1, x_2\rangle, x_3\rangle$.
3. $\langle x_1, x_2, \ldots, x_n\rangle := \langle \ldots \langle\langle x_1, x_2\rangle, x_3\rangle, \ldots, x_n\rangle$.



Definition 1.5 (Gödel Numbering of Finite Objects)

The *Gödel number* of a sequence of n-tuples of integers $\{a_1, a_2, \ldots, a_n\}$ is defined as

$$a = p_1^{a_1+1} p_2^{a_2+1} \cdots p_k^{a_k+1}$$

where p_i is the ith prime number. Given a we can effectively recover the prime power $(a)_i = a_i + 1$. This coding is injective but not surjective on $\mathbb{N}$.

Numbering Turing programs P_e

Definition 1.6

Since each Turing program is a finite set of quintuples, we can list all Turing programs in such a way that for any program we can effectively find its number on the list, and conversely.

- Let P_e be the Turing program with code number e in this coding, also called the Göodel number or index e.
- Let $\Phi_e^{(n)}$ be the partial function of n variables computed by P_e. Let Φ_e denote $\Phi_e^{(1)}$.

Lemma 1.7 (Padding Lemma)

For each Turing machine M there are infinitely many Turing machines $M_1, M_2, \ldots$ such that each M_i computes the same partial function as M.

Numbering Turing programs P_e

Fact 1.8

There is no effective enumeration of all total computable functions.

Proof.

Suppose $\{\Psi_e\}_{e\in\omega}$ is an effective of all total computable functions. Let

$$\psi(x) = \Psi_x(x) + 1.$$

Then ψ is a total computable function. So there exist i such that $\psi = \Psi_i$. Then

$$\Psi_i(i) = \psi(i) = \Psi_i(i) + 1.$$

It is a contradiction! □

Enumeration Theorem and universal machine

Theorem 1.9 (Enumeration Theorem)

There is a computable function $\Psi(e,x)$ *such that* $\Psi(e,x) = \Phi_e(x)$. *And there is some* i *such that* $\Phi_i(e,x) = \Psi(e,x)$.

Proof. (informal).

We construct a Turing machine as follows.

- Given the pair of inputs (e,x) convert e to the Turing program P_e.
- Simulate the action of P_e on input x.
- The simulation begins with state q_1 on the leftmost symbol of x in standard input form.
- If the simulation is in state q_j reading symbol s_k, then M searches through the tuples in P_e until it finds the first of the form (q_j, s_k, q, t, X) and then performs the indicated action on the simulation tape.

□

A Turing machine $M(e,x)$ which computes $\Psi(e,x)$ is called a *universal Turing machine*.

s-m-n Theorem

Theorem 1.10 (s-m-n Theorem)

There is an injective computable function $s(x,y)$ such that for all x, y, and z,

$$\Phi_{s(x,y)}(z) = \Phi_x(y,z).$$

Proof. (informal).

- We defines an procedure to obtain Turing program $M_{x,y}(z)$ with inputs x and y.
  - Given x and y, on input z first obtains program P_x, and then applies P_x to input (y,z).
- This procedure defines a computable function. There is a Turing machine index $s(x,y)$ such that $\Phi_{s(x,y)}(z) = M_{x,y}(z)$.
- By the Padding Lemma, we can choose $s(x,y)$ such that $s(x,y)$ in strictly increasing in $\langle x,y\rangle$. Then $s(x,y)$ is injective.

□

Recursion Theorem

Theorem 1.11 (Recursion Theorem)

For every computable function f there is some n, called a fixed point of f, such that

$$\Phi_n = \Phi_{f(n)}.$$

Proof.

- Define a partial computable function

$$\Psi(y,z) = \begin{cases} \Phi_{f(\Phi_y(y))}(z) & \text{if } \Phi_y(y)\downarrow; \\ \uparrow & \text{o.w.} \end{cases}$$

- By s-m-n Theorem, there exists a computable function $s(x,y)$ s.t. $\Phi_{s(x,y)}(z) = \Phi_x(y,z)$.
- Let e be in index of Ψ. Then $\Psi(y,z) = \Phi_e(y,z) = \Phi_{s(e,y)}(z)$.
- As $s(e,y)$ is total computable, let m be an index of $s(e,y)$, i.e., $\Phi_m(y) = s(e,y)$.
- Let $y = m$, then $\Phi_m(m)\downarrow$ and $\Psi(m,z) = \Phi_{f(\Phi_m(m))}(z)$.
- On the other hand, $\Psi(m,z) = \Phi_{s(e,m)}(z) = \Phi_{\Phi_m(m)}(z)$.
- Let $n = \Phi_m(m)$, then $\Phi_n(z) = \Phi_{f(n)}(z)$.

□