Computation and Logic: I

Chapter 1 Introduction to theory of computation

Kazuyuki Tanaka

BIMSA

November 3, 2022

北京雁栖湖
应用数学研究院
YANQI LAKE BEIJING INSTITUTE OF
MATHEMATICAL SCIENCES AND APPLICATIONS

Logic and Computation I

- **Part 1. Introduction to Theory of Computation**
- **Part 2. Propositional Logic and Computational Complexity**
- **Part 3. First Order Logic and Decision Problems**

Part 1. Schedule

- Oct.27, (1) Automata and monoids
- Nov. 1, (2) Turing machines
- Nov. 3, (3) Computable functions and primitive recursive functions
- Nov. 8, (4) Decidability and undecidability
- Nov.10, (5) Partial recursive functions and computable enumerable sets
- Nov.12, (6) Rice's theorem and many-one reducibility

Recap: TM and 0-type languages

- A deterministic Turing machine is almost like a DFA with a read-write head moving on two-way infinite tape.
- The language accepted by a Turing machine is called a 0-**type language**.
- A **multi-tape Turing machine** is introduced and its accepting language is shown to be 0-type.
- A **nondeterministic Turing machine** is introduced and its accepting language is shown to be 0-type.
- The class of 0-type languages is closed under $\cap, \cup, \cdot$ and * (but not complementation).
- A Turing machine defines a (partial) function if for a given input, the remaining string on the tape in a final state should be regarded as the output.
- A function $f : A \to \Omega^*$ $(A \subset \Omega^*)$ is **Turing definable** iff $\{u\sharp f(u) : u \in A\}$ is a 0-type langauge.

Computable functions and primitive recursive functions

- Let $\mathbb{N}$ be the set of all natural numbers. $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ is called a **number-theoretic function**.
- Turing definable function gives a mapping from strings to strings. It can be translated into a number-theoretic function.

Definition

A number-theoretic function $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ is **computable** if there is a Turing machine $\mathcal{M}$ accepts

$$1^{m_1}01^{m_2}0\cdots01^{m_k} := \underbrace{1\cdots1}_{m_1}0\underbrace{1\cdots1}_{m_2}0\cdots0\underbrace{1\cdots1}_{m_k}$$

and outputs

$$1^{f(m_1,\ldots,m_k)}.$$

We also say $\mathcal{M}$ **realizes** the function f.

By the last theorem of the last lecture, we have

$$f \text{ is computable} \Leftrightarrow \{1^{m_1}0\cdots01^{m_k}01^{f(m_1,\ldots,m_k)} : m_1,\ldots,m_k \in \mathbb{N}\}$$
$$\text{is a } 0\text{-type language on } \{0,1\}.$$

Example: Addition

Addition $+ : \mathbb{N}^2 \longrightarrow \mathbb{N}$ is computable.

It can be easily realized by a single tape Turing machine:

- the input is 1^m01^n,
- replace 0 with 1 and remove the rightmost 1 on the tape.

$$\overbrace{1\ 1\ 1\cdots1\ 1\ 1}^{m}\ 0\ \overbrace{1\ 1\ 1\cdots1\ 1\ 1}^{n}$$

$$\overbrace{1\ 1\ 1\cdots1\ 1\ 1}^{m}\ 1\ \overbrace{1\ 1\ 1\cdots1\ 1\ 1}^{n}$$

$$\underbrace{\overbrace{1\ 1\ 1\cdots1\ 1\ 1}^{m}\ 1\ \overbrace{1\ 1\ 1\cdots1\ 1}^{n-1}}_{m+n}\ B$$

Example: Multiplication

Multiplication $\cdot \;: \mathbb{N}^2 \longrightarrow \mathbb{N}$ is computable.

It can be realized by a 3-tape Turing machine:

- the input on the 1st tape is 1^m01^n, other two tapes are empty,
- then copy 1^m to the 2nd tape, copy 1^n to the 3rd tape, and make the 1st tape empty,
- repeat the following steps until the 3rd tape is empty:
  - remove the rightmost 1 on the 3rd tape and copy the content on the 2nd tape 1^m to the 1st tape right after the string already on the tape (if the 1st tape is empty, copy to any position)
- the output is 1^{mn}.

The 3rd tape works as a counter, computing how many times the TM copies the content on the 2nd tape to the 1st tape.

- Multiplication of natural numbers can be seen as a repetition of addition operation.
- Multiplication can be defined recursively:

$$\begin{cases} x \cdot 0 = 0, \\ x \cdot (y+1) = x \cdot y + x. \end{cases}$$

- More generally, the computable functions are closed under (primitive) recursive definition:

Lemma

If $g : \mathbb{N} \longrightarrow \mathbb{N}$, $h : \mathbb{N}^2 \longrightarrow \mathbb{N}$ are computable, a function $f : \mathbb{N}^2 \longrightarrow \mathbb{N}$ defined recursively as

$$\begin{cases} f(x,0) = g(x), \\ f(x,y+1) = h(x, f(x,y)) \end{cases}$$

is also computable.

Proof. To realize f, we construct a 3-tape Turing machine $\mathcal{M}$ as follows.

- The input on the 1st tape is 1^x01^y.
- Copy 1^x to the 2nd tape, 1^y to the 3rd and remain 1^x on the 1st.
- Carry out the computation of $g(x)$ on the 1st tape.
- Repeat as below:
  (1) If the 3rd tape is empty, $\mathcal{M}$ enters a final state;
  (2) Otherwise, $\mathcal{M}$ will remove the rightmost 1 on the 3rd tape,
  copy the content 1^x on the 2nd tape together with the separator 0 to the left of the current content 1^y on the 1st tape,
  carry out the computation of h on the fist tape. Go to (1).
- On the 1st tape, $\mathcal{M}$ computes
  $f(x,0) = g(x),\ f(x,1) = h(x, g(x)),\ \ldots,\ f(x,y) = h(x, f(x,y-1))$ in this order.
- Finally, $\mathcal{M}$ outputs $1^{f(x,y)}$.

□

Primitive recursive functions

- The computable functions defined from simple basic functions by primitive recursion (as in the above lemma) are called primitive recursive functions.
- Most of the number-theoretic functions used in ordinary mathematics are primitive recursion. But there exists a computable function which is not primitive recursive (ex. the Ackermann function).
- The primitive recursion functions are congenial to Hilbert's finitistism (supporting his formalist philosophy). But the exact definition of those functions were conceived in Gödel's proof of the incompleteness theorems.

Definition

The primitive recursive function is defined as below.

1. Constant 0, **successor function** $\mathrm{S}(x) = x + 1$, **projection** $\mathrm{P}_i^n(x_1, x_2, \ldots, x_n) = x_i$ $(1 \leq i \leq n)$ are primitive recursive functions.
2. **Composition**.
   If $g_i : \mathbb{N}^n \to \mathbb{N}$, $h : \mathbb{N}^m \to \mathbb{N}\,(1 \leq i \leq m)$ are primitive recursive functions, so is $f = h(g_1, \ldots, g_m) : \mathbb{N}^n \to \mathbb{N}$ defined as below:
   $$f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)).$$
3. **Primitive recursion**.
   If $g : \mathbb{N}^n \to \mathbb{N}$, $h : \mathbb{N}^{n+2} \to \mathbb{N}$ are primitive recursive functions, so is $f : \mathbb{N}^{n+1} \to \mathbb{N}$ defined as below:
   $$f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n),$$
   $$f(x_1, \ldots, x_n, y+1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)).$$

A primitive recursive function is a computable total function.

Lemma

Let $f(x_1, \ldots, x_n)$ be a primitive recursive n-ary function. Select n variable $y_{i_1}, \ldots, y_{i_n}$ (repetition is allowed) in a proper order from a list of m variables $y_1, \ldots, y_m$ and define a m-ary function

$$f'(y_1, \ldots, y_m) = f(y_{i_1}, \ldots, y_{i_n}).$$

f' is a primitive recursive function.

Proof.

- First, we treat the case when f is a constant function, using induction on m to show that m-ary f' is primitive recursive.
  - The basic case $m = 0$, f' is primitive recursive since $f'() = f()$.
  - Assume m-ary function $f_m(y_1, \cdots, y_m) = f()$ is primitive recursive.
    An $(m+1)$-ary function $f_{m+1}(y_1, \cdots, y_m, y_{m+1}) = f()$ is defined as below:

$$\begin{array}{rcl} f_{m+1}(y_1, \cdots, y_m, 0) & = & f_m(y_1, \cdots, y_m) \\ f_{m+1}(y_1, \cdots, y_m, z+1) & = & \mathrm{P}^{m+2}_{m+2}(y_1, \cdots, y_m, z, f_{m+1}(y_1, \cdots, y_m, z)). \end{array}$$

    Therefore $f_{m+1}(y_1, \cdots, y_m, y_{m+1})$ is also primitive recursive.
- Let n denote the arity of f and $n > 0$. f' is defined as:

$$f'(y_1, \cdots, y_m) = f(\mathrm{P}^m_{i_1}(y_1, \cdots, y_m), \cdots, \mathrm{P}^m_{i_n}(y_1, \cdots, y_m)).$$

Thus f' is primitive recursive. $\square$

Example

Let $f(x) = n$ be a primitive recursive function, e.g., if $n = 3$, then $f(x) = \mathrm{S}(\mathrm{S}(\mathrm{S}(\mathrm{Z}())))$.

Example

Predecessor function $\mathrm{M}(x) = x - 1$ $(x > 0)$, $\mathrm{M}(x) = 0$ $(x = 0)$.

$$\begin{cases} \mathrm{M}(0) = 0, \\ \mathrm{M}(x+1) = x = \mathrm{P}^2_1(x, \mathrm{M}(x)). \end{cases}$$

Example

Addition $\mathrm{plus}(x, y) = x + y$.

$$\begin{cases} \mathrm{plus}(x, 0) = x, \\ \mathrm{plus}(x, y+1) = \mathrm{S}(\mathrm{plus}(x, y)). \end{cases}$$

rewritten as,

$$\begin{cases} x + 0 = x, \\ x + (y+1) = \mathrm{S}(x + y). \end{cases}$$

Example

Subtraction $x \dot{-} y$.

$$\begin{cases} x \dot{-} 0 = x, \\ x \dot{-} (y+1) = \mathrm{M}(x \dot{-} y). \end{cases}$$

Homework 2.1

Prove $x \cdot y$, x^y, $x!$, $\max\{x, y\}$, $\min\{x, y\}$ are primitive recursive functions.

Homework 2.2

Let $f(x_1, \ldots, x_n, y)$ be a primitive recursive function. Prove the following functions are also primitive recursive.

$$F(x_1, \ldots, x_n, z) = \Sigma_{y<z} f(x_1, \ldots, x_n, y),$$

$$G(x_1, \ldots, x_n, z) = \Pi_{y<z} f(x_1, \ldots, x_n, y).$$

Definition

An n-ary relation $R \subset \mathbb{N}^n$ is called $\mathrm{primitive\ recursive}$, if its characteristic function $\chi_R : \mathbb{N}^n \to \{0, 1\}$ is primitive recursive

$$\chi_R(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } R(x_1, \ldots, x_n) \\ 0 & \text{otherwise} \end{cases}$$

Example

$x < y$ is primitive recursive. In fact,

$$\chi_<(x, y) = (y \dot{-} x) \dot{-} \mathrm{M}(y \dot{-} x).$$

Lemma

Given primitive recursive n-ary relation A, B,

$$\neg A,\ A \wedge B,\ A \vee B$$

are also primitive recursive.

Proof.

$$\chi_{\neg A} = 1 \dot{-} \chi_A$$

$$\chi_{A \wedge B} = \chi_A \cdot \chi_B$$

$$\chi_{A \vee B} = 1 \dot{-} \{(1 \dot{-} \chi_A) \cdot (1 \dot{-} \chi_B)\}$$

□

Definition by cases

Lemma

Given two primitive recursive n-ary functions g and h, and a primitive recursive n-ary relation R, f defined as follows is also primitive recursive,

$$f(x_1, \ldots, x_n) = \begin{cases} g(x_1, \ldots, x_n) & \text{if } R(x_1, \ldots, x_n) \\ h(x_1, \ldots, x_n) & \text{otherwise} \end{cases}$$

Example

$x = y$ is primitive recursive. Because $x = y \Leftrightarrow \neg(x < y) \wedge \neg(y < x)$.

Then, the following is obvious.

Lemma

The graph of a primitive recursive function is primitive recursive.

Homework 2.3

Prove that if $A(x_1, \ldots, x_n, y)$ is primitive recursive, $\forall y < z\ A(x_1, \ldots, x_n, y)$ and $\exists y < z\ A(x_1, \ldots, x_n, y)$ are also primitive recursive.

Example

$\mathrm{prime}(x) =$ "x is a prime number" is a primitive recursive relation. Actually,

$$\mathrm{prime}(x) \Leftrightarrow x > 1 \wedge \neg \exists y < x \exists z < x (y \cdot z = x).$$

Lemma

If $A(x_1, \ldots, x_n, y)$ is primitive recursive, the function $\mu y < zA$ satisfying the following condition is primitive recursive,

$$\mu y < zA(x_1, \ldots, x_n, y) = \min(\{y < z : A(x_1, \ldots, x_n, y)\} \cup \{z\}).$$

Proof.

$\mu y < zA = \Sigma_{w<z}\Pi_{y \leq w}\chi_{\neg A}$. $\square$

We can also prove that for a primitive recursive function $h(\vec{x})$, $\mu y < h(\vec{x})A(\vec{x}, y)$ is primitive recursive.

Example

Division $x/y = \mu z < x(x < y \cdot (z+1))$ is primitive recursive.

Example

Let $p(x) =$ "$(x+1)$th prime number ", that is ,

$$p(0) = 2, p(1) = 3, p(2) = 5, \ldots$$

Then, $p(x)$ is a primitive recursive function since it is defined as follows.

$$p(0) = 2, \quad p(x+1) = \mu y < p(x)! + 2 \ (p(x) < y \wedge \mathrm{prime}(y)).$$

Example

- A finite sequence of natural numbers $(x_0, \ldots, x_{n-1})$ can be represented by a single natural number x as follows,

$$x = p(0)^{x_0+1} \cdot p(1)^{x_1+1} \cdot \cdots \cdot p(n-1)^{x_{n-1}+1}$$

- Fixing n, such a mapping from $\mathbb{N}^n$ to $\mathbb{N}$ is a primitive recursive function.
- Conversely, for a natural number x, the function $c(x, i)$ takes the ith element x_i from x,

$$x_i = c(x, i) = \mu y < x \ (\neg \exists z < x \, (p(i)^{y+2} \cdot z = x)).$$

- The length of the sequence represented by x is

$$\mathrm{leng}(x) = \mu i < x \ (\neg \exists z < x \, (p(i) \cdot z = x)).$$

- Furthermore, we define a primitive recursive relation $\mathrm{Seq}(x)$ to denote that a natural number x is the code of such a sequence as follows:

$$\mathrm{Seq}(x) \Leftrightarrow \forall i < x \forall z < x \ (p(i) \cdot z = x \rightarrow i \leq \mathrm{leng}(x)).$$

Gödel number

Definition

Let Ω be a finite (or countably infinite) set of symbols, and an injection $\phi : \Omega \to \mathbb{N}$. For a string $s = a_0 \cdots a_{n-1}$, the following natural number $\psi(s)$ is called the **Gödel number** of s, denoted by $\ulcorner s \urcorner$.

$$\psi(s) = p(0)^{\phi(a_0)+1} \cdot p(1)^{\phi(a_1)+1} \cdot \cdots \cdot p(n-1)^{\phi(a_{n-1})+1}.$$

The mapping $\ulcorner\ \urcorner$ is an injection from the set of all symbols Ω^* to $\mathbb{N}$.

Example

Let $\Omega = \{0, 1, +, (,)\}$, $\phi(0) = 0$, $\phi(1) = 1$, $\phi(+) = 3$, $\phi(\,(\,) = 5$ and $\phi(\,)\,) = 6$.
Then,

$$\ulcorner (1+0)+1 \urcorner = 2^6 \cdot 3^2 \cdot 5^4 \cdot 7^1 \cdot 11^7 \cdot 13^4 \cdot 17^2$$

Homework 2.4

The symbol set Ω is the same as the example above. "Terms" are defined as below

(1) 0, 1 are terms.

(2) if s and t are terms, so is $(s+t)$.
e.g., $((1+0)+1)$ is a term, but $(1+0)+1$ is not a term.

Show that the predicate $\mathrm{Term}(x)$ expressing "x is the Gödel number of a term" is primitive recursive.

Recursive functions

Definition

A **recursive function** is defined as below.

1. **Constant** 0,
   Successor function $\mathrm{S}(x) = x + 1$,
   Projection $\mathrm{P}_i^n(x_1, x_2, \cdots, x_n) = x_i$ $(1 \leq i \leq n)$ are recursive functions.
2. **Composition**. The same as a primitive recursive function.
3. **Primitive recursion**. The same as a primitive recursive function.
4. **minimalization** (minimization).
   Let $g : \mathbb{N}^{n+1} \to \mathbb{N}$ be a recursive function satisfying that $\forall x_1 \cdots \forall x_n \exists y\, g(x_1, \cdots, x_n, y) = 0$. Then, the function $f : \mathbb{N}^n \to \mathbb{N}$ defined by

$$f(x_1, \cdots, x_n) = \mu y(g(x_1, \cdots, x_n, y) = 0)$$

   is recursive, where $\mu y(g(x_1, \cdots, x_n, y) = 0)$ denotes the smallest y such that $g(x_1, \cdots, x_n, y) = 0$,

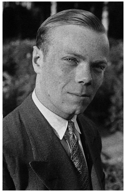
- Recursive functions are (total) computable functions, like primitive recursive functions.
- Condition 4 in the above definition (not in the definition of primitive recursive functions) is problematic sometimes, since it is often difficult to guarantee its totality condition $\forall x_1 \cdots \forall x_n \exists y \; g(x_1, \cdots, x_n, y) = 0$ in a formal system.
- E.g., the class of recursive functions allowed in Peano arithmetic does not match the class of recursive functions allowed in ZF set theory.
- A function defined by removing this totality condition is called **a partial recursive function**, and we will discuss it again later (Lecture 5 on Nov.10).

Summary

- f is computable iff $\{1^{m_1}0\cdots01^{m_k}01^{f(m_1,\ldots,m_k)} : m_1,\ldots,m_k \in \mathbb{N}\}$ is a 0-type language on $\{0,1\}$.
- Primitive recursive function (0, sucessor function, projection, closed under composition and primitive recursion)
- Recursive function (0, sucessor function, projection, closed under composition and primitive recursion, minimalization)

Further readings

N. Cutland. *Computability: An Introduction to Recursive Function Theory*, Cambridge University Press, 1st edition, 1980

Gödel and Herbrand (1933): partial recursive functions

Turing (1936): Turing machines

Church (1936): λ-calculus

Church, Kleene, and Turing proved that these three models of computation define the same classes of computable functions.

- Gödel and Herbrand (1933): partial recursive functions

  $\hookrightarrow$ a branch of mathematics called recursion theory

- Turing (1936): Turing machines

  $\hookrightarrow$ theory of computation

- Church (1936): λ-calculus

  $\hookrightarrow$ a branch of CS called functional programming

- many others...

Thank you for your attention!