Computation and Logic I

Chapter 1 Introduction to theory of computation

Kazuyuki Tanaka

BIMSA

September 19, 2024

北京雁栖湖
应用数学研究院
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 4. Modal logic**

Part 1. Schedule

- Sep.10, (1) Automata and monoids
- Sep.12, (2) Turing machines
- Sep.19, (3) Computable functions and primitive recursive functions
- Sep.24, (4) Decidability and undecidability
- Sep.26, (5) Partial recursive functions and computable enumerable sets
- Oct. 8, (6) Rice's theorem and many-one reducibility

Recap: TM and type-0 languages

- A **deterministic Turing machine** (TM) 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 **type-0 language**.
- A **multi-tape Turing machine** was introduced and its accepting language is shown to be type-0.
- A **nondeterministic Turing machine** was introduced and its accepting language is shown to be type-0.
- The class of type-0 languages is closed under $\cap, \cup, \cdot$ and * (but not complementation as shown later).

- 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.
- This is called a Turing definable function. Such a function is partially defined, since the TM does not always terminate.
- To make the output unique, we define the output of a (deterministic) TM as the string on the tape when the TM enters a final state for the first time, because it might enter a final state more than once.

Remark

- For a multitape TM and a nondeterministic TM, the output should be considered to be the output of equivalent single tape deterministic ones.

Theorem 1.16

Let $\sharp$ be a new symbols not included in Ω. The following are equivalent:
(1) A function $f : A \to \Omega^*$ $(A \subset \Omega^*)$ can be defined by a TM with output.
(2) $\{u\sharp f(u) : u \in A\}$ is a type-0 language.

Proof.
$(1) \Rightarrow (2)$.
Assume a partial function $f : \Omega^* \to \Omega^*$ is defined by a deterministic TM $\mathcal{M}$. We define a 2-tape $\mathcal{M}'$ which accepts $\{u\sharp f(u) : u \in A\}$ as follows:

- It checks whether a string on the 1st tape is in the form of $u\sharp v$. If not, then it stops in a non-final state.
- If so, $\mathcal{M}'$ copies u to the 2nd tape and simulates $\mathcal{M}$ on the 2nd tape.
- If $\mathcal{M}$ enters a final state, $\mathcal{M}'$ checks whether the string on the 2nd tape is the same as v on the 1st tape. If and only if it is the same, $\mathcal{M}'$ also enters a final state.

$(2) \Rightarrow (1)$.
Assume a TM $\mathcal{M}'$ that accepts $\{u\sharp f(u) : u \in A\}$. Next, we consider a nondeterministic $\mathcal{M}$ (with output).

- $\mathcal{M}$ has 2 tapes, one for input and the other for a working space.
- $\mathcal{M}$ non-deterministically produces a string $v \in \Omega^*$ on the 2nd tape.
- Write $\sharp$ after the input string u on the 1st tape, and copy v after $\sharp$. Then, mimic $\mathcal{M}'$ on the 1st tape.
- When it reaches a final state, it empties the 1st tape, copies the contents of the 2nd tape onto it, and then $\mathcal{M}$ enters a final state.
- The nondeterminism lies in writing an arbitrary string on the 2nd tape, which is equivalent to enumerating all the possible $f(u)$.

$\square$

- A Turing definable function is a mapping from strings to strings. But it can be translated into a (number-theoretic) function $f : \mathbb{N}^k \longrightarrow \mathbb{N}$.

Definition 1.17

A function $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ is **(Turing) computable** if there is a TM $\mathcal{M}$ 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, we have

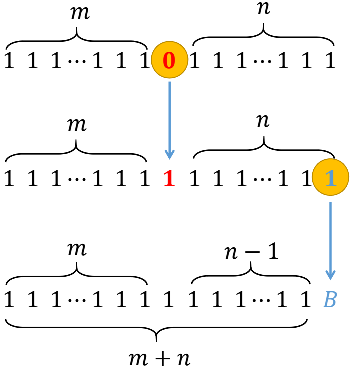
$$f \text{ is computable} \Leftrightarrow \{1^{m_1}0\cdots01^{m_k}01^{f(m_1,\ldots,m_k)} : m_1,\ldots,m_k \in \mathbb{N}\}$$
$$\text{is a type-}0\text{ language on } \{0,1\}.$$

Example 4: Addition

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

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

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



Example 5: Multiplication

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

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

- On the 1st tape, input is given as $1^m 0 1^n$, while the other 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 1^m on the 2nd tape 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 for computing how many times the TM copies the content on the 2nd tape to the 1st tape.

- Multiplication can be seen as a repetition of addition. In fact, multiplication can be defined recursively as follows:

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

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

Lemma 1.18

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 $f(x, y)$, 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 $f(x, 0) = g(x)$, $f(x, 1) = h(x, f(x, 0))$, $\ldots$, $f(x, y) = h(x, f(x, y - 1))$ 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 number-theoretic functions used in ordinary mathematics are primitive recursive. 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 1.19

The primitive recursive function is defined as below.

1. **Constant** 0, **successor function** $\mathrm{S}(x) = x + 1$, and **projection** $\mathrm{P}^n_i(x_1, x_2, \dots, x_n) = x_i$ $(1 \le i \le n)$ are primitive recursive functions.
2. **Composition**.
   If $g_i : \mathbb{N}^n \to \mathbb{N}$, $h : \mathbb{N}^m \to \mathbb{N}\,(1 \le i \le m)$ are primitive recursive functions, so is $f = h(g_1, \dots, g_m) : \mathbb{N}^n \to \mathbb{N}$ defined as below:
   $$f(x_1, \dots, x_n) = h(g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n)).$$
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:
   $$\begin{aligned} f(x_1, \dots, x_n, 0) &= g(x_1, \dots, x_n), \\ f(x_1, \dots, x_n, y+1) &= h(x_1, \dots, x_n, y, f(x_1, \dots, x_n, y)). \end{aligned}$$

The following is obvious from Lemma 1.18 and Definition 1.19.

Lemma 1.20

A primitive recursive function is a computable total function.

The following is also easy from Definition 1.19.

Lemma 1.21

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:

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

    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:

$$f'(y_1, \cdots, y_m) = f(\mathrm{P}^m_{i1}(y_1, \cdots, y_m), \cdots, \mathrm{P}^m_{in}(y_1, \cdots, y_m)).$$

Thus f' is primitive recursive. □

Example 6

A constant function $f(x) = n$ is a primitive recursive function, e.g., if $n = 3$,

$$f(x) = \mathrm{S}(\mathrm{S}(\mathrm{S}(\mathrm{Z}()))).$$

Example 7

The predecessor function $\mathrm{M}(x) = x - 1$ $(x > 0)$, with $\mathrm{M}(x) = 0$ $(x = 0)$, is a primitive recursive function, since

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

Example 8

Addition $\mathrm{plus}(x, y) = x + y$ is primitive recursive, since

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

or rewritten as

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

Example 9

Subtraction $x \dot{-} y$ is primitive recursive, since

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

Exercise 1.3.1

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

Exercise 1.3.2

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 1.22

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

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

Example 10

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

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

Lemma 1.23

Given primitive recursive n-ary relation A, B, then

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

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 1.24

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

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

Proof.

$f = g \cdot \chi_R + h \cdot \chi_{\neg R}$. □

Example 11

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

Then, the following is obvious.

Lemma 1.25

The graph of a primitive recursive function is primitive recursive.

Exercise 1.3.3

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 12

$\mathrm{prime}(x) =$ "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 1.26

If $A(x_1, \ldots, x_n, y)$ is primitive recursive, the function $\mu y < zA$ defined by 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.

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

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 13

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

Example 14

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.

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

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

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

Example 15

- Fixing n, a mapping from $(x_0, \ldots, x_{n-1}) \in \mathbb{N}^n$ to its sequence number $x \in \mathbb{N}$ is a primitive recursive function.
- Conversely, let $c(x, i)$ be a function taking the i-th element x_i from x. It is primitive recursive, since

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

- The length of a sequence x, denote $\mathrm{leng}(x)$, is primitive recursive, since

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

- Finally, we define a relation $\mathrm{Seq}(x)$ to mean that x is a sequence number. Then it is primitive recursive, since

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

Gödel numbers

Definition 1.27

Let Ω be a finite (or countably infinite) set of symbols with an injection $\phi : \Omega \to \mathbb{N}$. For a string $s = a_0 \cdots a_{n-1}$ from Ω, the number sequence of $(\phi(a_0) \cdots \phi(a_{n-1}))$, i.e.,

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

is called the **Gödel number** of s, denoted by $\ulcorner s \urcorner$.

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

Example 16

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

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

Exercise 1.3.3

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 1.28

The **recursive functions** are defined as follows

1. **Constant** 0, **Successor** $\mathrm{S}(x) = x + 1$, **Projections** $\mathrm{P}^n_i(x_1, x_2, \cdots, x_n) = x_i$ are recursive functions. (These basic functions are also primitive recursive.)
2. **Composition**. The same as a primitive recursive function.
3. **Primitive recursion**. The same as a primitive recursive function.
4. **minimalization** (or **minimization**).
   Let $g : \mathbb{N}^{n+1} \to \mathbb{N}$ be a recursive function such that $\forall x_1 \cdots \forall x_n \exists y\ g(x_1, \cdots, x_n, y) = 0$. Define a function $f : \mathbb{N}^n \to \mathbb{N}$ by

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

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

- Recursive functions are (total) computable functions, like primitive recursive functions.
- However, condition 4 in the above definition (not included 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 absolutely computable way, or in a rigid formal system.
- For instance, 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 later (in Lecture 5).

Summary

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

Further readings

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

Thank you for your attention!