Logic and Computation: I

Chapter 1 Introduction to theory of computation

Kazuyuki Tanaka

BIMSA

November 8, 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

Definition

The primitive recursive functions are 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 prim. rec. functions.
2. **Composition**.
   If $g_i (1 \leq i \leq m)$, h are prim. rec. functions, so is $f = h(g_1, \ldots, g_m)$ defined by:
   $$f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)).$$
3. **Primitive recursion**.
   If g, h are prim. rec. functions, so is f defined by:
   $$f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n),$$
   $$f(x_1, \ldots, x_n, y+1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)).$$

A primitive recursive function is a computable total function.

Example

$x + y$, $x \dot{-} y$, $x \cdot y$, x/y, x^y, $x!$, $\max\{x, y\}$, $\min\{x, y\}$ are primitive recursive functions.

Example

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

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

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

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

Definition

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

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

Primitive recursive relations are closed under Boolean operations and bounded quantifiers.

Example: $x < y$ is primitive recursive

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

Example: $x = y$, prime(x) are primitive recursive

$$x = y \Leftrightarrow \neg(x < y) \wedge \neg(y < x).$$

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

Recursive functions

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

Definition

The set of all recursive functions is the smallest class that contains the constant 0, successor function, projection, and closed under composition, primitive recursion and minimalization.

A recursive function is a computable total function, and vice versa.

Computability and Incomputability (Uncomputability)

Today

- We will only consider a deterministic single-tape Turing machine on $\Omega = \{0, 1, \mathrm{B}\}$.
- We will introduce a **P**rogramming **L**anguage, called **TPL**, that has an instruction for each operation of **T**uring machine.
- Any Turing machine can be emulated by a TPL program on a unique Turing machine (called universal Turing machine).
- Finally, we will prove the existence of an incomputable (non-computable) set K.

Definition (The instructions of Programming language TPL)

instructions	(code $\sharp$, the corresponding TM operations)
halt	(code 0, enter a final state)
moveright	(code 1, the head move to right by one cell)
moveleft	(code 2, the head move to left by one cell)
write 0	(code 3, write "0" on the tape)
write 1	(code 4, write "1" on the tape)
write B	(code 5, write "B" On the tape)
goto l	(code $6 + 3l$, jump to the lth instruction)
if 0 **then goto** l	(code $7 + 3l$, if TM reads 0, jump to the lth instruction)
if 1 **then goto** l	(code $8 + 3l$, if TM reads 1, jump to the lth instruction)

Definition

A **program** of TPL is a list of instructions separated by ";".

For readability, a line number is added at each instruction.
In the instruction "**goto** l", l corresponds to such a line number.

An example of TPL program $\mathcal{P}_0$

0: **if** 1 **then goto** 2;
1: **goto** 1;
2: **moveright**;
3: **if** 1 **then goto** 1;
4: **if** 0 **then goto** 6;
5: **halt**;
6: **moveright**;
7: **goto** 0

The left program intends to accept the language $1(01)^*$

Definition (TM $\mathcal{M}_\mathcal{P}$ realizes TPL program $\mathcal{P}$)

Let $\mathcal{P}$ be be a TPL program. We define a (deterministic) Turing machine $\mathcal{M}_\mathcal{P} = (Q, \Omega, \delta, q_0, F)$ realizes $\mathcal{P}$. $Q = \{0, 1, \dots, n-1\}$ is the set of line numbers of $\mathcal{P}$. $\Omega = \{0, 1, \mathrm{B}\}$. $q_0 = 0$, $F = \{$a line number of **halt**$\}$. The transition function $\delta : Q \times \Omega \to \Omega \times \{L, R, N\} \times Q$ is defined as follows.

l: **halt**,	$\delta(l, x) = (x, N, l)$,
l: **moveright**,	$\delta(l, x) = (x, R, l+1)$,
l: **moveleft**,	$\delta(l, x) = (x, L, l+1)$,
l: **write** $?$,	$\delta(l, x) = (?, N, l+1)$, for $? = 0, 1, \mathrm{B}$,
l: **goto** k,	$\delta(l, x) = (x, N, k)$,
l: **if** $?$ **then goto** k,	$\delta(l, ?) = (?, N, k)$ and
	$\delta(l, y) = (y, N, l+1)$ for $y \neq ?$.

The language accepted by TPL $\mathcal{P}$ is the language accepted by the associated Turing machine $\mathcal{M}_\mathcal{P}$. The partial function $f : \Omega^* \to \Omega^*$ defined by $\mathcal{P}$ is a function defined by $\mathcal{M}_\mathcal{P}$.

Example: Program $\mathcal{P}_0 \Rightarrow$ TM $\mathcal{M}_{\mathcal{P}_0}$

We define a (deterministic) Turing machine $\mathcal{M}_{\mathcal{P}_0} = (Q, \Omega, \delta, q_0, F)$, where $Q = \{0, 1, \ldots, 7\}$, $\Omega = \{0, 1, \mathrm{B}\}$, $q_0 = 0$, $F = \{5\}$, and
δ is defined as follows: for any $x \in \Omega$,

0: **if** 1 **then goto** 2;	$\delta(0,1) = (1, N, 2)$, $\delta(l, y) = (y, N, 1)$ for $y \neq 1$
1: **goto** 1;	$\delta(1, x) = (x, N, 1)$
2: **moveright**;	$\delta(2, x) = (x, R, 3)$
3: **if** 1 **then goto** 1;	$\delta(3,1) = (1, N, 1)$, $\delta(3, y) = (y, N, 4)$ for $y \neq B$
4: **if** 0 **then goto** 6;	$\delta(4,0) = (0, N, 6)$, $\delta(4, y) = (y, N, 5)$ for $y \neq 1$
5: **halt**;	$\delta(5, x) = (x, N, 5)$
6: **moveright**;	$\delta(6, x) = (x, R, 7)$
7: **goto** 0	$\delta(7, x) = (x, N, 0)$

A TPL program $\mathcal{P}_1$

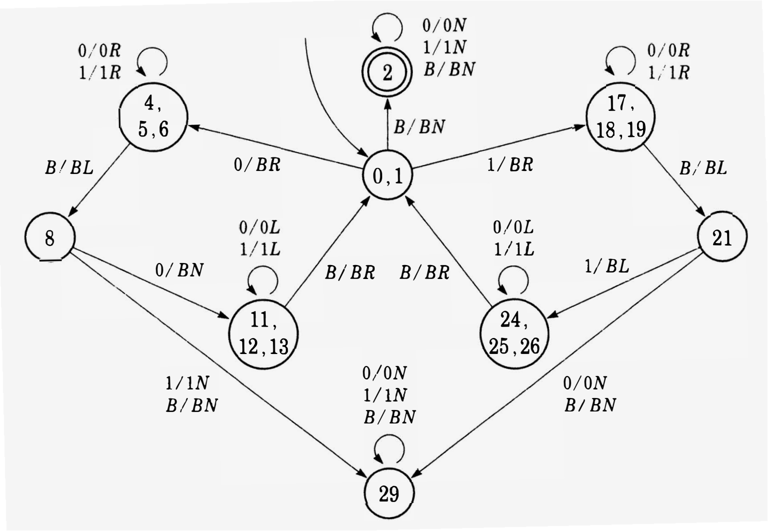
0: **if** 0 **then goto** 3;
1: **if** 1 **then goto** 16;
2: **halt**;

3: **write** B;
4: **moveright**;
5: **if** 0 **then goto** 4;
6: **if** 1 **then goto** 4;
7: **moveleft**;
8: **if** 0 **goto** 10;
9: **goto** 29;
10: **write** B;
11: **moveleft**;
12: **if** 0 **then goto** 11;
13: **if** 1 **then goto** 11;
14: **moveright**;
15: **goto** 0;

16: **write** B;
17: **moveright**;
18: **if** 0 **then goto** 17;
19: **if** 1 **then goto** 17;
20: **moveleft**;
21: **if** 1 **goto** 23;
22: **goto** 29;
23: **write** B;
24: **moveleft**;
25: **if** 0 **then goto** 24;
26: **if** 1 **then goto** 24;
27: **moveright**;
28: **goto** 0;

29: **goto** 29

TM $\mathcal{M}_1$ for $\mathcal{P}_1$ and $L(\mathcal{M}_1) = \{ww^R : w \in \{0,1\}^*\}$



Theorem

For any Turing machine $\mathcal{M}$, there exists a TPL program $\mathcal{P}$ such that $L(\mathcal{M}) = L(\mathcal{M}_{\mathcal{P}})$.

Proof. (Exercise.)

A program $\mathcal{P}$ is a sequence of instructions with codes $c_0, c_1, \ldots, c_l$.

Example

0: **if** 1 **then goto** 2;	Code $c_0 = 8 + 3 \cdot 2 = 14$
1: **goto** 1;	Code $c_1 = 6 + 3 \cdot 1 = 9$
2: **moveright**;	Code $c_2 = 1$
$\vdots$	$\vdots$

$\mathcal{P}$ can be represented by a sequence $1^{c_0}0 \cdots 01^{c_l}$ on $\{0,1\}^*$.

The Gödel number of a program $\ulcorner \mathcal{P} \urcorner$ is

$$p(0)^{c_0+1} \cdot p(1)^{c_1+1} \cdot \cdots \cdot p(l)^{c_l+1}.$$

According to the previous theorem, for any TM $\mathcal{M}$, there is a TPL program $\mathcal{P}_\mathcal{M}$.
The Gödel number $\ulcorner \mathcal{P}_\mathcal{M} \urcorner$ is called the index (or code) of TM $\mathcal{M}$.

Partial computable functions

- The definition of computable functions in Lecture 3 can be applied to **partial functions**. Namely, the partial function $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ is **computable** if $\{1^{m_1}0 \cdots 01^{m_k}01^{f(m_1,\ldots,m_k)} : m_1, \ldots, m_k \in \mathbb{N}\}$ is a 0-type language.
- Then, the partial function f realized by $\mathcal{M}$ with index e is represented by $\{e\}^k$ (or simply $\{e\}$) (called **Kleene's bracket notation**).
- When e is not an index of TM, $\{e\}$ is regarded as a partial function with empty domain.

Any TM can be formulated as a TPL Program. So, we can construct a "Universal Turing Machine" as a TPL interpreter. More generally, we have the following theorem.

Theorem (Enumeration theorem)

For any $n \geq 0$, there exists a natural number e_n such that for any $d, x_1, \ldots, x_n$,

$$\{e_n\}^{n+1}(d, x_1, \ldots, x_n) \sim \{d\}^n(x_1, \ldots, x_n).$$

$f(x_1, \ldots, x_n) \sim g(x_1, \ldots, x_n)$ means either both sides are not defined or they are defined with the same value.

This theorem affirms the existence of a universal TM with index e_n that is able to mimic any TM with index d.

Proof. We will construct a universal Turing machine $\mathcal{M}$ with index e_n.

- $\mathcal{M}$ has one input tape and two working tapes.
- Let $1^d 0 1^{x_1} 0 \cdots 0 1^{x_n}$ be an input on the first tape.
- Let the index part 1^d represent the program
  {the instruction of code c_0;
  the instruction of code c_1;
  $\cdots$;
  the instruction of code c_l}.
- Write $1^{c_0} 0 \cdots 0 1^{c_l}$ on the 2nd tape and remove $1^d 0$ on the 1st tape.
- Execute the instructions on the 2nd tape sequentially, rewriting the string on the 1st tape with the help of the 3rd tape.

Proof.(Continued)

- The 3rd tape will be used to find the next executable operation when the **goto** instruction or **if** ? **then goto** instruction is executed on the 2nd tape.
  - 1^{6+3l} sandwiched between two 0's means **goto** l, so the next executable instruction is given by the sequence of 1's between the l-th 0 and the $l+1$-th 0. To find it on the 2nd tape, we need to store the number of 0's counted from the left to the end of the string.
- Instructions other than **goto** and **if** ? **then goto** can be easily executed, and finding the next executable instruction is also obvious.
- When **halt** instruction is executed, $\mathcal{M}$ enters a final state.
- At that time, $1^{\{d\}^n(x_1,\ldots,x_n)}$ is written on the 1st tape.

$\square$

Definition

A set $X \subset \mathbb{N}^n$ is called computably enumerable, CE for short, if

$$\{1^{x_1}0 \cdots 01^{x_n} : (x_1, \ldots, x_n) \in X\}$$

is a 0-type language.

In other words, $X \subset \mathbb{N}^n$ is CE iff it is the domain of some partial computable function. Other equivalent defintions will be given in the next lecture.

Now, we denote K as a set of natural numbers defined as

$$\mathrm{K} := \{e : e \in \mathrm{dom}(\{e\}^1)\} = \{e : (e,e) \in \mathrm{dom}(\{e_1\}^2)\}.$$

where e_1 is the code of the universal TM in the previous theorem. We call K the **halting problem**. Strictly speaking, this is a special kind of halting problem, and the general case K_0 will be given later.

Theorem (Turing)

K is a CE set and its complement $\mathbb{N} - \mathrm{K}$ is not CE.

Proof.

- To show K is CE.
  We construct a TM accepting $\{1^e : e \in \mathrm{K}\}$ as follows.
  - For input 1^e, it rewrites as $1^e 0 1^e$ on the tape.
  - Then, the TM mimics the universal TM that realizes $\{e_1\}^2$.
  - This TM enters the final state, when $(e, e) \in \mathrm{dom}(\{e_1\}^2)$, that is, if and only if $e \in \mathrm{K}$.
- By contradiction, assume that $\mathbb{N} - \mathrm{K}$ is a CE set.
  Assume a TM that accepts $\{1^e : e \notin K\}$, with code d. At this time,

$$d \in \mathrm{K} \Leftrightarrow d \in \mathrm{dom}(\{d\}^1) \Leftrightarrow d \in \{e : e \notin \mathrm{K}\} \Leftrightarrow d \notin \mathrm{K}$$

Therefore, either $d \in \mathrm{K}$ or $d \notin \mathrm{K}$ leads to a contradiction.

Definition

A set $X \subset \mathbb{N}^n$ is **computable** (or **recursive** , **decidable**) if both X and its complement are CE.

- K is an incomputable CE set.
- "A set $X \subset \mathbb{N}^n$ is computable" is equivalent to its characteristic function is computable,

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

  - Because if we have partial computable functions f and g with $\mathrm{dom}(f) = X$ and $\mathrm{dom}(g) = \mathbb{N}^n - X$, then for any input $1^{x_1}0 \cdots 01^{x_n}$, the computations for f, g can be done in parallel and decide the output to 1 or 0 depending on which one stops first. Such computations are totally defined since it always terminates.

- In general, if a function $f(x)$ has a finite value at $x = n$, we write $f(n)\downarrow$. That is

$$f(n)\downarrow \Leftrightarrow n \in \mathrm{dom}(f).$$

Then we also write K as

$$\mathrm{K} = \{e : \{e\}(e)\downarrow\}$$

Problem

Show that the following two sets are noncomputable CE set.

$$\mathrm{K}_0 = \{(x, e) : \{e\}(x)\downarrow\},$$

$$\mathrm{K}_1 = \{e : \mathrm{dom}(\{e\}) \neq \emptyset\}.$$

- K_0 is the original **halting problem**: given a program and input, decide when the machine will halt.
- However, since we use the special halting problem K more frequently, we refer K_0 as the "membership decision problem (MEM)".

Summary

- Any Turing machine can be emulated by a TPL program on the so-called universal Turing machine.
- Enumeration theorem
- A set $X \subset \mathbb{N}^n$ is CE if $\{1^{x_1}0 \cdots 01^{x_n} : (x_1, \ldots, x_n) \in X\}$ is a 0-type language.
- X is computable if both X and X^c are CE.
- All computable sets are CE. But the reverse is not true.

Further readings

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

Thank you for your attention!