Note on Recursion theory(Draft)

from Zero to Hero

Xin Chen

Master Student at the SYSU
Website
GitHub
Laste update: May 26, 2023

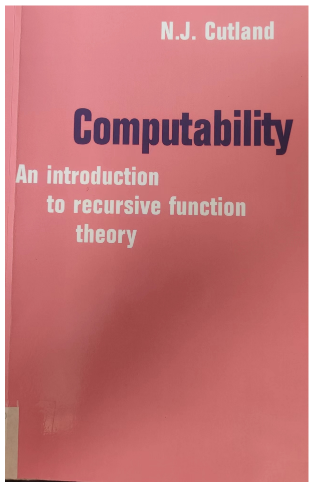
chenxin_hello@outlook.com

Contents

Notes on Recursion Theory (draft)

Chen Xin Last update: May 26, 2023

Course: Recursion theory
Textbook: N.J. Cutland, *Computability: An introduction to recursive function theory*, CUP, 1980 (the Pink Book)



Lecturer: Prof. Yongfeng Yuan
Time and Venue: 14:20 - 17:10 Fri. / A556-Haiqin-6, Zhuhai campus , SYSU

Other References:

1. https://github.com/blargoner/math-computability-cutland

Cheat sheets:

Table 1. some computable functions

ddddd	fffff

Table 2. some undecidable problems

undecidable problems		other equivalent forms and comments	page
	'ϕ_x is total'		p.90
	'$x \in W_x$'	'$\phi_x(x)$ is defined'; '$P_x(x)\downarrow$'; '$\psi_U(x,x)$ is defined'	p.101
Halting problem:	'$P_x(y)\downarrow$'	'$\phi_x(y)$ is defined' '$y \in W_x$'	p.102
0-function problem	'$\phi_x = \mathbf{0}$'		p.103
function equal problem	'$\phi_x = \phi_y$'		p.104
Input problem:	'$c \in W_x$'	'$P_x(c)\downarrow$', '$c \in Dom(\phi_x)$'	p.104
Output problem:	'$c \in E_x$'	'$c \in Ram(\phi_x)$'	p.104
Rice's Theorem	'$\phi_x \in \mathcal{R}$'	$\mathscr{C}_1 \supset \mathcal{R} \neq \emptyset$	p.105

annotation

the page in Pink Book

Prerequisites

Some basic concepts and definitions of sets and functions:

- $A \subseteq B$、$A \subset B$ 、$A \cup B$、$A \cap B$ (subset, proper subset, union, intersection)
- setminus: $A \setminus B := \{x \mid x \in A, x \notin B\}$; complement :$\bar{A} := \mathbb{N} \setminus A$
- Cartesian product: $A \times B := \{(x, y) \mid x \in A, y \in B\}$
- $A^n := A \times \cdots \times A$ (n times)

  ....

- function f: $(x, y), (x, z) \in f \Rightarrow y = z$
- $Dom(f) := \{x \mid f(x) \text{ is defined }\}$
- $Ran(f) := \{f(x) \mid x \in Dom(f)\}$
- f is a function from A to B if $Dom(f) \subseteq A, Ran(f) \subseteq B$.
- $f : A \to B$ iff f is a function from A to B and $Dom(f) = A$
- f is *injective* (injection): $\forall x, y \in Dom(f), x \neq y \Rightarrow f(x) \neq f(y)$.

Remark 只有单射才能讨论逆函数 inverse f^{-1}

- f is *surjective* (surjection):
- *bijective* (bijection) = injective + surjective
- restriction $f|X$ (or $f\lfloor X$):
- *composition*: $f \circ g$; whose domain is $\{x \mid x \in Dom(g) \;\&\; g(x) \in Dom(f)\}$
  $(f \circ g)(x) := f(g(x))$

- $\alpha(\bar{x}) \simeq \beta(\bar{x})$: for any $\bar{x}$, $\alpha(\bar{x}), \beta(\bar{x})$ are either both defined, or both undefined, and if defined they are equal.

. . . .

- Functions of natural numbers
  - *total function*: whose domain is the whole of $\mathbb{N}^n$
  - *partial function*: whose domain is not necessarily the whole of $\mathbb{N}^n$.
  - *zero function*: $\mathbf{0}\colon \mathbb{N} \to \mathbb{N}$ such that $\mathbf{0}(x) = 0$.
  - $\mathbf{m}\colon \mathbb{N} \to \mathbb{N}$ such that $\mathbf{m}(x) = m$.

. . . .

- equivalence relation, equivalence class
- *partial order*: irreflexivity(禁自反) + transitivity(传递)

可计算函数类有多大?
= 递归函数类

CHAPTER 1

Computable functions

Target What is an algorithm or effective procedure?
What is an idealised computer?

Keywords 内容...

Table 1.1. Content of Chapter 1

Section 1.1

Algorithms and Effective Procedures

Generally, an **algorithm** or **effective procedure** is a mechanical rule, or automatic method, or programme for performing some mathematical operation. For example:

- given n, finding the nth prime number;
- finding the highest common factor of two numbers (Euclidean algorithm);
- given two number x, y deciding whether x is a multiple of y;
- ......

Note 1.1 effectively calculable = algorithmically computable = effectively computable = computable

two features of effective procedure:

1. such procedure is carried out in a sequence of stages or steps (finite time),
2. any output should emerge after a finite number of steps.

Idealised computer: inputs and outputs will be restricted to natural numbers, this is not a significant restriction, since operations involving other kinds of object can be *coded* as operations on natural numbers.

Section 1.2

URM: the unlimited register machine

Mathematical ideal computer: URM (unlimited register machine, 1963)[1]

- URM has an infinite number of **registers** $R_1, R_2, R_3, \ldots$, each contains a natural number at any moment. Fixed a moment, r_i is the number in R_i.
- the contents of the registers may be altered by apply certain *instructions*.

[1] 无限寄存器

- **program**: a finite list of instructions.

Four kinds **instructions** (three *arithmetic* instructions):

1. **Zero instruction** $Z(n)$: to change the contents of R_n to 0
   $0 \to R_n$
2. **Successor instruction** $S(n)$: to increase the number in R_n by 1
   $r_n + 1 \to R_n$
3. **Transfer instruction** $T(m, n)$: replace the contents of R_n by the number r_m in R_m
   $r_m \to R_n$
4. **Jump instruction** $J(m, n, q)$:
   - if $r_m = r_n$, then the URM proceeds to the qth instruction of P.
   - if $r_m \neq r_n$, then the URM proceeds to the next instruction.

   If P has less than q instructions, which means the jump is impossible, then the URM stops operation.

Table 1.2. four kinds instructions

name	Instruction	Response of the URM
zero	$Z(n)$	replace r_n by 0.
successor	$S(n)$	add 1 to r_n.
transfer	$T(m, n)$	replace r_n by r_m.
jump	$J(m, n, q)$	if $r_m = r_n$ jump to the qth instruction; otherwise go on to the next instruction.

Computation:

initial configuration
computation *stop*: there is not next instruction
final configuration

Notation

1. $P(a_1, a_2, a_3, \dots)$: the computation under P with *initial configuration* $a_1, a_2, a_3, \dots$.
2. $P(a_1, a_2, a_3, \dots) \downarrow$: the computation $P(a_1, a_2, a_3, \dots)$ eventually stops. (**converge**) [2]
3. $P(a_1, a_2, a_3, \dots) \uparrow$: the computation $P(a_1, a_2, a_3, \dots)$ never stops. (**diverge**) [3]

A computation that stops is said to *converge*, and never stops is said to *diverge*.

[2] 收敛

[3] 发散

Section 1.3

URM computable functions

Definition 1.2 (**Computable functions**) Let f be a *partial function* from $\mathbb{N}^n$ to $\mathbb{N}$, and P is a program.

1. $P(a_1, a_2, \dots, a_n)$ converges to b if $P(a_1, a_2, \dots, a_n)\downarrow$, and b is in R_1 in the final configuration. Notation: $P(a_1, a_2, \dots, a_n)\downarrow b$.
2. P URM **computes** f: $P(a_1, a_2, \dots, a_n)\downarrow d \quad \Leftrightarrow \quad (a_1, a_2, \dots, a_n) \in Dom(f)$ & $f(a_1, a_2, \dots, a_n) = b$.
3. f is URM **computable** if there is a program P that URM computes f.

We will use the term *computable* to mean URM computable.

Notation The class of all computable function is denoted by $\mathscr{C}$.
The class of all n-ary computable functions by $\mathscr{C}_n$.

Example 1.3 Following functions are all computable (the second line is a program that computes it) [4]:

- $x + y$
  $\langle J(3,2,5); S(1); S(3); J(1,1,1)\rangle$
- $x \dot{-} 1 = \begin{cases} x-1 & x > 0 \\ 0 & x = 0 \end{cases}$
  $\langle J(1,4,9); S(3); J(1,3,7); S(2); S(3); J(1,1,3); T(2,1)\rangle$
- $f(x) = \begin{cases} \dfrac{1}{2}x & x \text{ is even} \\ \text{undefined} & x \text{ is odd} \end{cases}$
  $\langle J(1,2,6); S(3); S(2), S(2), J(1,1,1), T(3,1)\rangle$

[4] *p.17-21* Pink Book

Given a program P, there is a *unique* n-ary functions that P computes, denoted by $f_P^{(n)}$, clearly

$$f_P^{(n)}(a_1, \dots, a_n) = \begin{cases} \text{the unique } b \text{ such that } P(a_1, \dots, a_n)\downarrow b & P(a_1, \dots, a_n)\downarrow \\ \text{undefined} & P(a_1, \dots, a_n)\uparrow \end{cases}$$

Obviously, a given computable function can be computed by many different program, infinitely many indeed.

Section 1.4

Decidable predicates and problems

Definition 1.4 (**Characteristic function**) Suppose $M(x_1, x_2, \dots, x_n)$ is a n-ary predicate on $\mathbb{N}$. Let $\bar{x} = (x_1, x_2, \dots, x_n)$, then the **characteristic function** [5] $c_M(\bar{x})$ of $M(\bar{x})$ is given by

$$c_M(\bar{x}) = \begin{cases} 1 & \text{if } M(\bar{x}) \text{ holds,} \\ 0 & \text{if } M(\bar{x}) \text{ doesn't holds,} \end{cases}$$

[5] 刻画函数

Definition 1.5 (**Decidability**) A predicate $M(\bar{x})$ on $\mathbb{N}$ is **decidable** if its characteristic function c_M is computable; $M(\bar{x})$ is **undecidable** if $M(\bar{x})$ is not decidable.

Example 1.6 Following predicates are all decidable:

- '$x \neq y$'
- '$x = 0$'
- 'x is a multiple of y'

Remark ☞ when discussing *decidability* we are always concerned with the computability (or non-computability) of **total** functions

SECTION 1.5

Computability on other domains

Our definition of computability and decidability applies only on natural number so far. These notions are easily extended to other kinds of object, e.g. polynomials, matrices, etc. by means of *coding*.

A **coding** of a domain D of objects is an explicit and effective *injection*[6] $\alpha\colon D \to \mathbb{N}$. We say that $d \in D$ is **coded** by $\alpha(d)$.

[6] *hence D must be a countable set.*

Suppose f is a function from D to D, then f is coded by then function f^* from $\mathbb{N}$ to $\mathbb{N}$ that maps the code of an object $d \in Dom(f)$ to the code of $f(d)$, that is,

$$f^* = \alpha \circ (f \circ \alpha^{-1})$$

f is **computable** if f^* is computable.

Thinking ? 1.7 不可数集上的可计算问题是否没有意义？或者这压根就不能成为一个问题？
上面所说的可计算函数都是一个相同集合上面的，即同一个论域；那么跨论域的可计算函数怎么定义呢？

SECTION 1.6

Selected exercises of Chapter 1

Exercise (2.2) Carry out the computation under the program of example 2.1 with initial configuration $\boxed{8}\boxed{4}\boxed{2}\boxed{0}\boxed{0}\boxed{\cdots}$. p.14

PROOF The program of example 2.1 is :

$$\begin{array}{llll} I_1 & J(1,2,6) & I_4 & J(1,2,6) \\ I_2 & S(2) & I_5 & J(1,1,2) \\ I_3 & S(3) & I_6 & T(3,1) \end{array}$$

$$\begin{tikzcd}
N \arrow[r, "f^*"] & N' \arrow[d, "\alpha^{-1}"] \\
D \arrow[u, "\alpha"] \arrow[r, "f"] & D'
\end{tikzcd}$$

where $N = N' = \mathbb{N}, D = D'$.

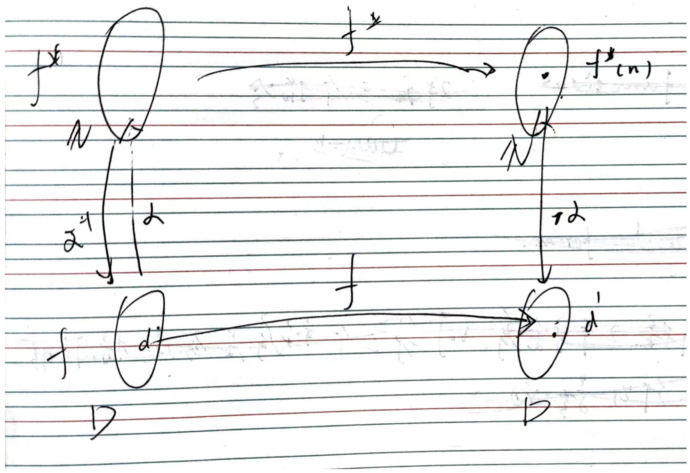


Figure 1.1. coding

	R_1	R_2	R_3	R_4	R_5		next instruction
Initial configuration	8	4	2	0	0	...	I_1
	8	4	2	0	0	...	I_2 (since $r_1 \neq r_2$)
	8	5	2	0	0	...	I_3
	8	5	3	0	0	...	I_4
	8	5	3	0	0	...	I_5 (since $r_1 \neq r_2$)
	8	5	3	0	0	...	I_2 (since $r_1 = r_1$)
	8	6	3	0	0	...	I_3
	8	6	4	0	0	...	I_4
	8	6	4	0	0	...	I_5 (since $r_1 \neq r_2$)
	8	6	4	0	0	...	I_2 (since $r_1 = r_1$)
	8	7	4	0	0	...	I_3
	8	7	5	0	0	...	I_4
	8	7	5	0	0	...	I_5 (since $r_1 \neq r_2$)
	8	7	5	0	0	...	I_2 (since $r_1 = r_1$)
	8	8	5	0	0	...	I_3
	8	8	6	0	0	...	I_4
	8	8	6	0	0	...	I_6 (since $r_1 = r_2$)
Final configuration	6	8	6	0	0	...	I_7: STOP

■

Exercise (3.3-1-(b),(c)(d)) Show that the following functions are computable by devising programs that will compute them. p.21

(b) $f(x) = 5$

(c) $f(x,y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}$

(d) $f(x,y) = \begin{cases} 0 & \text{if } x \leq y \\ 1 & \text{if } x > y \end{cases}$

Proof (b) Following program computes $f(x) = 5$:

$$I_1: S(2) \quad I_2: S(2) \quad I_3: S(2) \quad I_4: S(2) \quad I_5: S(2) \quad I_6: T(2,1)$$

(c) Following program computes $f(x,y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}$:

$$I_1: J(1,2,5) \quad I_2: Z(1) \quad I_3: S(1) \quad I_4: J(1,1,6) \quad I_5: Z(1)$$

(d) Following program computes $f(x,y) = \begin{cases} 0 & \text{if } x \leq y \\ 1 & \text{if } x > y \end{cases}$:

$$\begin{array}{ll} I_1 & J(1,2,7) \\ I_2 & S(3) \\ I_3 & S(1) \\ I_4 & J(2,3,9) \\ I_5 & J(1,2,7) \\ I_6 & J(1,1,2) \\ I_7 & Z(1) \\ I_8 & J(1,1,11) \\ I_9 & Z(1) \\ I_{10} & Z(0) \end{array}$$

■

Exercise (4.3 (c)) Show that the following predicates are decidable. p.23

(c) 'x is even'

Proof It is sufficient to show that the characteristic function of 'x is even'is computable, namely to show that the function

$$C_M(x) = \begin{cases} 1 & x \text{ is even} \\ 0 & x \text{ is not even} \end{cases}$$

is computable.

Following program computes function C_M:

$$
\begin{array}{ll}
I_1 & J(1,2,8) \\
I_2 & S(3) \\
I_3 & S(2) \\
I_4 & S(2) \\
I_5 & J(1,2,8) \\
I_6 & J(1,3,11) \\
I_7 & J(1,1,2) \\
I_8 & Z(1) \\
I_9 & S(1) \\
I_{10} & J(1,1,12) \\
I_{11} & Z(0)
\end{array}
$$

■

Generating computable functions

CHAPTER
2

Target From some computable functions to generate other computable functions, hence we can without writing a *program* each time to decide whether a function is computable.

Keywords 内容...

Table 2.1. Content of Chapter 2

Section 2.1

Three basic computable functions

There are three **basic computable functions** (with a program which computes it):

1. the *zero function*[7]: $\mathbf{0}$;
   $Z(1)$
2. the *successor function*[8]: $x+1$;
   $S(1)$
3. the *projection function*[9]: $U_i^n(x_1, x_2, \ldots, x_n) = x_i$, where $1 \le n, 1 \le i \le n$.
   $T(i, 1)$

These functions just correspond to three kinds *arithmetic instructions* (see page 5) for URM .

[7] 零函数

[8] 后继函数

[9] 投影函数

Section 2.2

Joining programs together [未完成]

Definition 2.1 (**Standard form**) A program $P = I_1, \ldots, I_s$ is in **standard form**, if for each *jump instruction* $J(m, n, q)$ in P we have $q \le s+1$.

It is easy to convert any program P into standard form.

Lemma 2.2 For any program P there is a program P^* in standard form, and these two programs are equivalent. In particular, for any $a_1, \ldots, a_n$ and b:

$$P(a_1, \ldots, a_n)\downarrow b \;\Leftrightarrow\; P^*(a_1, \ldots, a_n)\downarrow b,$$

and hence $f_P^{(n)} = f_{P^*}^{(n)}$ for all $n > 0$.

PROOF Suppose $P = I_1, \dots, I_s$, then define $P^* = I_1^*, \dots, I_s^*$ by

$$I_k^* = \begin{cases} I_k & \text{if } I_k \text{ is not a jump instruction} \\ I_k & I_k = J(m,n,q), q \le s+1 \\ J(m,n,s+1) & I_k = J(m,n,q), q > s+1 \end{cases}$$

where $1 \le k \le s$. ■

Definition 2.3 (**Join** or **Concatenation**) Let P and Q be programs of lengths s and t in standard form. The **join** or **concatenation** P and Q, notation PQ, is the program

$$I_1, I_2, \dots, I_s, I_{s+1}, \dots, I_{s+t}$$

where $P = I_1, I_2, \dots, I_s$ and $I_{s+1}, \dots, I_{s+t}$ are the instructions of Q with each *jump* $J(m,n,q)$ replaced by $J(m,n,s+q)$.

Note 2.4

- PQ 就是把 P 运算完的结果 (final configuration) 作为 Q 的初始配置 (initial configuration);
- 如果只拼接两个程序，那么第二个程序是不是标准形式对运算结果没有任何影响。

SECTION 2.3

Substitution

Target $\mathscr{C}$ (the class of all computable functions) is closed under the operation of *substitution*.

Theorem 2.5 (**Substitution Theorem**) Suppose $f(y_1, \dots, y_k)$ and $g_1(\bar{x}), \dots, g_k(\bar{x})$ are computable functions, where $\bar{x} = (x_1, \dots, x_n)$. Then the function $h(\bar{x})$ given by

$$h(\bar{x}) \simeq f(g_1(\bar{x}), \dots, g_k(\bar{x}))$$

is computable.

PROOF 内容... ■

SECTION 2.4

Recursion

$$\varphi$$

SECTION 2.5

Minimalisation

φ

SECTION 2.6

Selected exercises of Chapter 2

Exercise (3.4.1-(a),(b)) Without writing any programs, show that for every $m \in \mathbb{N}$ the following functions are computable: p.32 in Pink Book

(a) $\mathbf{m}$ (recall that $\mathbf{m}(x) = m$ for all $x \in \mathbb{N}$),

(b) mx.

PROOF (a) Let $\mathbf{m}(x) = \underbrace{ss\cdots s}_{m}(\mathbf{0}(x))$

Since *zero function* and *successor function* are both computable, so is $\mathbf{m}$ (by induction on m and *Theorem 3.1* (p.29)).

(b) Let

$$mx = \begin{cases} 0 & \text{if } m = 0, \\ \underbrace{x + x + \cdots + x}_{m} & \text{otherwise,} \end{cases}$$

Then we by induction on m to show this function is computable.

Base case: if $m = 0$, then $mx = 0 = \mathbf{0}(x)$, since the *zero function* $\mathbf{0}$ is computable, so is mx.

Induction step: if $m > 0$, suppose mx is computable, then $(m+1)x = mx + x$ by the definition. Since $(y + z)$ is computable, by substituting mx for y, and x for z, according *Theorem 3.1* (p29), we have that $(mx + 1)$ is computable. ■

Exercise 3.4.3 Suppose the $g(x)$ is a total computable function. Show that the predicate $M(x, y)$ given by p.32 in Pink Book

$$M(x, y) \equiv g(x) = y$$

is decidable.

PROOF ....method 1 ....

Considering $f(x, y) = \begin{cases} 0 & x = y \\ 1 & x \neq y \end{cases}$ which is computable. Then we substitute $g(x)$ into $f(x, y)$ by

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

Let $C_M(x, y) = \overline{sg}(f(g(x), y))$, since $\overline{sg}, f(x, y)$ and $g(x)$ are computable, so is $C_M(x, y)$ by *Theorem 3.1* (p29).

..................................method 2 ....

It suffices to show that the characteristic function of $M(x, y)$ is computable, namely to show

$$c_M(x,y) = \begin{cases} 1 & g(x) = y \\ 0 & g(x) \neq y \end{cases}$$

is computable.
Since $g(x)$ is total computable, suppose program P computes $g(x)$, let P^* be a standard form of P and P^* has u instructions. Now constructing a program Q as follows:

$$\begin{array}{ll} \vdots & P^* \\ I_{u+1} & Z(2) \\ I_{u+2} & S(2) \\ \vdots & \vdots \\ I_{u+y+1} & S(2) \\ I_{u+y+2} & J(1,2,u+y+6) \\ I_{u+y+3} & Z(1) \\ I_{u+y+4} & S(1) \\ I_{u+y+5} & J(1,1,u+y+7) \\ I_{u+y+6} & S(0) \end{array}$$

And it is easy to show that Q computes $c_M(x,y)$. ∎

下面是第 4 章的题啦

Exercise (4.6.2) Devise Turing machines that will Turing computes the functions
(a) $x \dot{-} 1$ (b) $2x$

p.57 in Pink Book

PROOF (a)
$x \dot{-} 1 = \begin{cases} x-1 & x > 0 \\ 0 & x = 0 \end{cases}$
The Turing machine with following instructions will computes this function:

$$q_1 1 B q_2$$

(b) $2x$
The Turing machine with following instructions will computes this function: ∎

SECTION 2.7

Church's thesis

- Church, Turing and Markov: the class of functions they had defined coincides with the informally defined class of effectively computable functions.
- **Church's thesis (Church-Turing thesis)**

  the intuitively and informally defined class of effectively computable partial functions coincides exactly with the class $\mathscr{C}$ of URM-computable functions.
- Church's thesis is not a theorem, but a claim or belief which must be substantiated by evidence.

The evidence for Church's thesis:

1. the fundamental result:

   many independent proposals for a precise formulation of the intuitive idea led to the same class of functions $\mathscr{C}$.

2. A vast collection of effectively computable functions has been shown explicitly belong to $\mathscr{C}$.

   No one has ever found a function that would be accepted as computable in the informal sense, but doesn't belong to $\mathscr{C}$.

   [like 'Completeness': each effectively computable functions is $\mathscr{C}$'s member]

3. A program P on the URM to compute a function is clearly an example of an algorithm; thus, directly from the definition of the class $\mathscr{C}$, we see that

   all functions in $\mathscr{C}$ are computable in the informal sense.

   [like 'Soundness': each $\mathscr{C}$'s member is effectively computable function]

Hence most mathematicians are led to accept Church's thesis.

$\Rightarrow\Rightarrow$ Suppose that we have a *informally described algorithm* for a function f, how can we prove that f is URM-computable? (three methods) $\Leftarrow\Leftarrow$

- write a program that URM-computes f; or
- showing f belongs to $\mathscr{C}, \mathscr{R}$ or $\mathscr{T}\mathscr{R}$. or
- give an informal proof that the given informal algorithm is indeed an algorithm that serves to compute f.

  Then appeal to Church's thesis and conclude immediately that f is URM-computable (proof by Church's thesis).

Church's thesis not only keeps proofs shorter, but also prevents the main idea of a proof or construction from being obscured by a mass of technical details.

2023.04.14 hw

Exercise **7.2.1** Suppose $f(x)$ and $g(x)$ are effectively computable functions. Prove, using Church's thesis, that the function h given by (p.71)

$$h(x) = \begin{cases} x & \text{if } x \in Dom(f) \cap Dom(g) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Proof An algorithm for h can be described in terms of given algorithms of f and g as follows: ' Given x, star the algorithms for computing $f(x)$ and $g(x)$ simultaneously. If both computations terminate, then stop and set $h(x) = x$. Otherwise, continue indefinitely. ' ■

Exercise **7.2.2** Suppose that f is a total unary computable function. Prove, by Church's thesis, (p.71) that the following function h is URM-computable.

$$h(x) = \begin{cases} 1 & \text{if } x \in Ran(f) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Proof An algorithm for h can be described in terms of given algorithms of f as follows:
输入 x，从 0 开始计算 f：如果 $f(0) = x$ 则停止；否则判断 $f(1) = x$。
一致重复这个过程。如果有一个数 z 使得 $f(z) = x$，则停机并让 $h(x) = 1$. 否则一直不停机。 ∎

CHAPTER 3

Other approaches to computability: Church's thesis

Target This chapter consider two questions:

1. How do the many different approaches to the characterisation of computability compare with each other, and in particular with URM-computability.

2.

Section 3.1

Other approaches

Section 3.2

Partial recursive functions (Gödel-Kleene)

Section 3.3

Primitive recursive functions

Section 3.4

Turing computability

tape is infinite in both directions (本书考虑的是"双向无限图灵机"，而一般更常见的是"单向无限图灵机"，但是二者是等价的)

B: notation for blank (assume every *alphabets* contain B)

Program Q: (图灵机的程序没有顺序之分，只是一有穷 的指令 的集合)

three kinds of instruction:

1. $q_i s_j s_k q_l$

2. $q_i s_j R q_l$

3. $q_i s_j L q_l$

A Turing machine **terminates** only when it is in a state q_i and reading s_j but there is no $q_i s_j \nabla q_l$ in the program Q; that is, there is no quadruple in Q that specifies what to do next (but it is possible that this never happens).

Subsection 3.4.1
Turing-computable functions

Notice: the functions we considering are all over ω.
alphabet: $\{1, B\}$

Definition 3.1 (Turing computable) A *partial* function is **Turing computable** it there is a Turing machine that computes it.
The class of all Turing computable partial function is denoted $\mathscr{TC}$.

CHAPTER 4

Numbering computable functions

Target **Key facts** will be established:

- the set $\mathscr{P}$ of all programs (URM-program) is *effectively denumerable.*
  that is there is an effective coding of programs by $\mathbb{N}$.
- the class $\mathscr{C}$ of all computable functions is *denumerable.*
  hence there are many functions that are not computable.

Section 4.1

Numbering programs

Definition 4.1 (可枚举、枚举、能行可枚举)

- A set X is **denumerable** (可枚举的), if there is a *bijection* $f\colon X \to \mathbb{N}$.

  (note: **countable** means finite or denumerable; for infinite sets, countable means the same as denumerable)
  denumerable = countably infinite
- An **enumeration** (枚举) of a set X is a *surjection* $g\colon \mathbb{N} \to X$; this is often represented by
  $X = \{x_0, x_1, \dots\}$, where $x_n = g(n)$.

  This is an enumeration without *repetitions* if g is *injective.*
- Let X be a set of finite objects (but X itself may be infinite). Then X is **effectively denumerable** (能行可枚举的) if there is a *bijection* $f\colon X \to \mathbb{N}$ such that both f and f^{-1} are *effectively computable functions.*

  (这里的 f 是跨论域的)

【Say a function is **effectively computable**, if....】

猜想（直观上成立，但还没有看到有关证明）

If f is computable and bijective, then f^{-1} also computable.

enumeration: 枚举，名词;
denumerable: 可枚举的，形容词

Theorem 4.2 The following sets are effectively denumerable.
(a) $\mathbb{N} \times \mathbb{N}$ (b) $\mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+$
(c) $\bigcup_{k>0} \mathbb{N}^k$, the set of all finite sequences of natural numbers

PROOF (a) $\mathbb{N} \times \mathbb{N}$
method-1 (p73):
Let $\pi\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is given by (需要数论中的知识)

$$\pi(m, m) = 2^m(2n + 1) - 1$$

It is clear that π is computable. And π is a bijection:

- for injective:
- for surjective:

π^{-1} is given by

$$\pi^{-1}(x) = (\pi_1(x), \pi_2(x))$$

where

$$\pi_1(x) = (x+1)_1 \qquad \pi_2(x) = \frac{1}{2}\left(\frac{x+1}{2^{\pi_1(x)}} - 1\right).$$

It is clear that π_1 and π_2 are computable, hence also π^{-1}.
这里对函数 π_1 做些说明。$(n)_1$ 是一个数，其值等同于将 n 这个数进行质因数分解（prime factorization，任何自然数的质因数分解都是唯一的，这称为**算术基本定理**）后 2 的幂。之所以 $(n)_1$ 中有下标 1，是因为 2 是第一个质数（注意一般规定 1 不是质数）。
比如，$5 = 2^0 \times 5$ ，因此 $(5)_1 = 0$ ，进而 $\pi_1(4) = (4+1)_1 = 0$ ；而 $\pi_1(5) = (5+1)_1$，因为 $6 = 2 \times 3$ ，所以 $\pi_1(5) = 1$。类似地，$\pi_1(7) = (8)_1 = 3$，$\pi_1(9) = (10)_1 = 1$。
π_1, π_2 函数都是从 π 函数中提炼出来的。
....
method-2: use **diagonal-like/zig-zag argument** (which mentioned in class by Prof. Yuan).
Intuition: any element belongs to $\mathbb{N} \times \mathbb{N}$ can be listed at following net:
按照图中箭头从右下角开始数，则可以走遍 $\mathbb{N} \times \mathbb{N}$ 中所有的元素。
Recursively define a function g on $\mathbb{N}$ as follows:

$$\begin{aligned} g(0) &= 0 \\ g(k) &= g(k-1) + k \ \ (k > 0) \end{aligned}$$

$g(x)$ 表示 $(x, 0)$ 将被映射为 $\mathbb{N}$ 中的元素 $g(x)$。
Let $f\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ given by

$$f(m, n) = g(m+n) - n$$

f is injective: 【】【】【】
f is surjective: 非形式的论证 f 为满射：对于任意 $x \in \mathbb{N}$，从该表格的 $(0, 0)$ 开始按箭头方向走 x 步，若在点 (m, n) 停下，则 (m, n) 即为 x 的原像。按这种方式对于所有 $\mathbb{N}$ 中的元素总能为其找到 $\mathbb{N} \times \mathbb{N}$ 中的原像。
Clearly f is computable.
For f^{-1} is computable. (要用到极小化算子来论证)
【zig-zag 论证是一种直观且普适的方法，以后对于类似的问题都可以采用这种证明策略】
....
method-3 (from the slide of Prof. Yuan which is similar to **method-2**)

(b) $\mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+$

Using the function π in (a), a bijection $\zeta\colon \mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}$ is given by

$$\zeta(m, n, q) = \pi(\pi(m-1, n-1), q-1).$$

Then

$$\zeta^{-1} = [\pi_1(\pi_1(x)) + 1, \pi_2(\pi_1(x)) + 1, \pi_2(x) + 1].$$

Since π, π_1, π_2 are effectively computable, then so are ζ and ζ^{-1}.

(c) $\bigcup_{k>0} \mathbb{N}^k$

A bijection $\tau\colon \bigcup_{k>0} \mathbb{N}^k \to \mathbb{N}$ is given by

$$\begin{aligned}\tau(a_1, \ldots, a_k) = {} & 2^{a_1} + 2^{a_1+a_2+1} + 2^{a_1+a_2+a_3+2} + \cdots \\ & + 2^{a_1+a_2+\cdots+a_k+k-1} - 1.\end{aligned}$$

Clearly τ is effectively computable.
For bijective:
Fact: every natural number has a unique expression as a binary decimal.
Thus given a x we can effectively find unique number $k \geq 1$ and $0 \leq b_1 < b_2 < \cdots < b_k$ such that

$$x + 1 = 2^{b_1} + 2^{b_2} + \cdots + 2^{b_k}.$$

Then we obtain

$$\tau^{-1}(x) = (a_1, \ldots, a_k)$$

where $a_1 = b_1$ and $a_{i+1} = b_{i+1} - b_i - 1$ $(1 \leq i < k)$.
Moreover, τ^{-1} is effectively computable. ■

To sum up we have six functions from above theorem, that is,

1. $\pi\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

   $\pi(m, n) = 2^m(2n+1) - 1$

   $\pi^{-1}(x) = (\pi_1(x), \pi_2(x))$ where $\pi_1(x) = (x+1)_1$ $\quad \pi_2(x) = \frac{1}{2}(\frac{x+1}{2^{\pi_1(x)}} - 1)$.

2. $\zeta\colon \mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}$

   $\zeta(m, n, q) = \pi(\pi(m-1, n-1), q-1)$

   $\zeta^{-1} = [\pi_1(\pi_1(x)) + 1, \pi_2(\pi_1(x)) + 1, \pi_2(x) + 1]$.

3. $\tau\colon \bigcup_{k>0} \mathbb{N}^k \to \mathbb{N}$

   $$\begin{aligned}\tau(a_1, \ldots, a_k) = {} & 2^{a_1} + 2^{a_1+a_2+1} + 2^{a_1+a_2+a_3+2} + \cdots \\ & + 2^{a_1+a_2+\cdots+a_k+k-1} - 1.\end{aligned}$$

   $\tau^{-1}(x) = (a_1, \ldots, a_k)$ where $a_1 = b_1$ and $a_{i+1} = b_{i+1} - b_i - 1$ $(1 \leq i < k)$.

We will use these functions to coding programs later.
Some notations:

- $\mathscr{I}$: the set of all URM instructions. (I 的花体)
- $\mathscr{P}$: the set of all programs.

$\mathscr{I}$ 是个无穷集，且由四个无穷的子集组成: Z-指令集、S-指令集、T-指令集以及 J-指令集。显然 Z-指令有无穷多个，比如 $Z(1), Z(2), Z(3)\ldots$（注意不会有 $Z(0)$ 指令，因为寄存器是从 1 开始排列的）。

现在如果想把 $\mathscr{I}$ 编码到 $\mathbb{N}$ 中，就像要让无穷多个新旅客住进“希尔伯特旅馆”一样。

希尔伯特旅馆

dddd

Theorem 4.3 $\mathscr{I}$ is effectively denumerable.

Proof There are four kinds of instruction: Z, S, T, J.
Define $\beta\colon \mathscr{I} \to \mathbb{N}$ as follows

$$\begin{aligned}\beta(Z(n)) &= 4(n-1)\\ \beta(S(n)) &= 4(n-1)+1\\ \beta(T(m,n)) &= 4\pi(m-1,n-1)+2\\ \beta(J(m,n,q)) &= 4\zeta(m,n,q)+3\end{aligned}$$

Clearly β is effectively computable and surjective.
To find $\beta^{-1}(x)$:
first find u, r such that $x = 4u + r$ with $0 \le r < 4$.
The value of r indicates which kind of instruction $\beta^{-1}(x)$ is.
From u we can find the particular instruction of that kind. Specifically:
Hence β^{-1} is also effectively computable. ■

$\mathscr{P}$ 是 $\mathscr{I}$ 中所有有穷指令序列的集合（注意 URM 程序的指令是有序列的，而图灵机的则没有），因此，如果想证明 ¶ 是能行可枚举的，和证明 $\bigcup_{k>0}\mathbb{N}^k$ 是能行可枚举的方法一样。

(编码函数不是唯一的，现实中可能是用质数给程序来编码的)

Theorem 4.4 $\mathscr{P}$ is effectively denumerable.

Proof Define an explicit bijection $\gamma\colon \mathscr{P} \to \mathbb{N}$ as follows,
using the function τ and β:
If $P = I_1, I_2, \ldots, I_s$, them

$$\gamma(P) = \gamma(I_1, I_2, \ldots, I_s) = \tau(\beta(I_1), \beta(I_2), \ldots, \beta(I_s)).$$

Since τ and β are bijections, so is γ.
The fact that τ, β and their inverses are effectively computable ensures that γ and γ^{-1} are also effectively computable. ■

For each program P, the number $\gamma(P)$ is called the **code number** of P, or the **Gödel number** of P (Gödel who first exploited the idea of coding non-numerical object by numbers in his famous paper [Gödel 1931]).

We define

$$P_n = \gamma^{-1}(n) = \text{ the program with code number } n$$

and say P_n is the n-th program.

☞ Note that: if $m \neq n$, then P_m differs from P_m, although they may compute the same functions.

An important result: γ and γ^{-1} are effectively computable, that is:

- Given a particular program P, we can effectively find the code number of $\gamma(P)$.
- Given a particular number n, we can effectively find the program $P_n = \gamma^{-1}(n)$.

Example 4.5

(a) Let $P = \langle T(1,3), S(4), Z(6)\rangle$ be a program, we can find the code number of P:

$$\begin{aligned}\gamma(P) &= \gamma(T(1,3), S(4), Z(6)) \\ &= \tau(\beta(T(1,3)), \beta(S(4)), \beta(Z(6)))\end{aligned}$$

$\beta(T(1,3)) = 4\pi(1-1, 3-1) + 2 = 4 \times (2^0(2\times 2+1) - 1) + 2 = 18;$
$\beta(S(4)) = 4 \times (4-1) + 1 = 13$
$\beta(Z(6)) = 4 \times (6-1) = 20$
Thus, $\gamma(P) = \tau(18, 13, 20) = 2^{18} + 2^{32} + 2^{53} - 1 = 9007203549970431$. (一般只要算到 2 的指数就可以了)

....

(b) Let $n = 4127$, then to find P_{4127}.
$4127 = 2^5 + 2^{12} - 1$, hence P_{4127} has two instructions I_1, I_2.
$\tau^{-1}(4127) = (5, 12-5-1) = (5, 6)$
$\beta(I_1) = 5 = 4 \times 1 + 1$
$\beta(I_2) = 6 = 4 \times 1 + 2$
Thus, I_1 is S-instruction, and $\beta^{-1}(5) = S(1+1) = S(2)$;
I_2 is T-instruction, and $\beta^{-1}(6) = T(\pi_1(1)+1, \pi_2(1)+1) = T(1+1, 0+1) = T(2,1)$
(since $\pi_1(1) = (2)_1 = 1$ while $\pi_2(1) = 0.5 \times (\frac{1+1}{2^{\pi_1(1)}} - 1) = 0$)
Therefore, $P_{4127} = \langle S(2), T(2,1)\rangle$

Exercise 1.6 (c)-(d) p.76 Find
(c) the code number of the following program: $P = T(3,4), S(3), Z(1)$.

$$\begin{aligned}\gamma(P) &= \gamma(T(3,4), S(3), Z(1)) \\ &= \tau(\beta(T(3,4)), \beta(S(3)), \beta(Z(1)))\end{aligned}$$

$$\begin{aligned}\beta(T(3,4)) &= 4\pi(3-1, 4-1) + 2 \\ &= 4 \times \pi(2,3) + 2 \\ &= 4 \times (2^2 \cdot (2\times 3+1) - 1) + 2 \\ &= 4 \times 27 + 2 = 110\end{aligned}$$

$\beta(S(3)) = 4 \times (3-1) + 1 = 9$
$\beta(Z(1)) = 4 \times (1-1) = 0$
$\gamma(P) = \tau(110, 9, 0) = 2^{110} + 2^{120} + 2^{121} - 1$

....

(d) P_{100}
$100 + 1 = 2^0 + 2^2 + 2^5 + 2^6 = 2^{b_1} + 2^{b_2} + 2^{b_3} + 2^{b_4}$
hence,
$a_1 = b_1 = 0$
$a_2 = b_2 - b_1 - 1 = 2 - 0 - 1 = 1$
$a_3 = b_3 - b_2 - 1 = 5 - 2 - 1 = 2$
$a_4 = b_4 - b_3 = 6 - 5 - 1 = 0$
then
$\tau^{-1}(100) = (0, 1, 2, 0)$
$\beta^{-1}(0) = Z(0+1) = Z(1)$
$\beta^{-1}(1) = S(0+1) = S(1)$
$\beta^{-1}(2) = T(\pi_1(0)+1, \pi_2(0)+1) = T(1,1)$ (since $\pi_1(0) = \pi_2(0) = 0$)
Therefore, $P_{100} = \langle \beta^{-1}(0), \beta^{-1}(1), \beta^{-1}(2), \beta^{-1}(0)\rangle = \langle Z(1), S(1), T(1,1), Z(1)\rangle$.

SECTION 4.2

Numbering computable functions

可计算函数有可数无穷多个

Definition 4.6 For each $a \in \mathbb{N}$ and $n \geq 1$:

- $\phi_a^{(n)}$ = the n-ary function computed by P_a.
  $= f_{P_a}^{(n)}$
- $W_a^{(n)} = \{(x_1, \ldots, x_n) \mid P_a(x_1, \ldots, x_n)\downarrow\}$ = domain of $\phi_a^{(n)}$
- $E_a^{(n)}$ = range of $\phi_a^{(n)}$

where

$$f_{P_a}^{(n)}(x_1, \ldots, x_n) = \begin{cases} \text{the unique } \ b \text{ such that } P_a(x_1, \ldots, x_n)\downarrow b, & P_a(x_1, \ldots, x_n)\downarrow \\ \text{undefined}, & P_a(x_1, \ldots, x_n)\uparrow \end{cases}$$

For convenience, we write ϕ_a for $\phi_a^{(1)}$, W_a for $W_a^{(1)}$, and E_a for $E_a^{(1)}$.

如果不限制元数，则一个程序可以计算的函数有无穷多个，但是只要元数限定了，一个程序可以计算的 n 元函数是唯一的。因此上述定义是良定义的。

Example 4.7 if $a = 1427$, from previous example we know that $P_{4127} = S(2), T(2,1)$. Hence for unary function ϕ_{1427}, we have :

- $\phi_{1427}(x) = 1$ for all $x \in \mathbb{N}$
- $W_{4127} = \mathbb{N}$
- $E_{4127} = \{1\}$

For more general case, that n-ary function $\phi_{1427}^{(n)}$ $(n > 1)$, we have:

- $\phi_{1427}^{(n)}(x_1, x_2, \ldots, x_n) = x_2 + 1$
- $W_{4127}^{n} = \mathbb{N}^n$
- $E_{4127}^{(n)} = \mathbb{N}^+$

Let $f = \phi_a$, we say a is an **index** for f, each computable function has *infinitely many indices*, since there are infinitely many different programs that compute a same function.

Let $\mathscr{C}_n$ indicates the class of n-ary computable functions.

Theorem 4.8 $\mathscr{C}_n$ is denumerable, in other word, $\mathscr{C}_n$ countably infinite.

PROOF We have to show that there is a bijection $f\colon \mathscr{C}_n \to \mathbb{N}$. ■

SECTION 4.3

The Diagonal-method

Exercise 3.2: 1,3 (p80)

3.2-1 Suppose $f(x, y)$ is a total computable function. For each m, let g_m be the computable function given by

$$g_m(y) = f(m, y).$$

Construct a total computable function h such that for each m $h \neq g_m$.

PROOF | 内容... ■

3.2-3 Let $f\colon \mathbb{N} \to \mathbb{N}$ be partial and $m \in \mathbb{N}$. Construct a non-computable function g such that

$$g(x) \simeq f(x) \quad \text{for} \quad x \leq m.$$

PROOF | 内容... ■

SECTION 4.4

The s-m-n theorem

Suppose $f(x, y)$ is a computable function (may partial), then for each fixed value a of x, f gives rise to a unary computable function g_a, where

$$g_a(y) \simeq f(a, y).$$

Since g_a is computable, then it has an index e (not unique) such that

$$f(a, y) \simeq \phi_e(y).$$

The *s-m-n theorem* show that such an index e can be obtained effectively from a.

Theorem 4.9 (The **s-m-n theorem**, simple form) Suppose $f(x, y)$ is computable. There is a *total* computable function $k(x)$ such that

$$f(x, y) \simeq \phi_{k(x)}(y).$$

PROOF | 内容... ■

Theorem 4.10 (The **s-m-n theorem**) For each $m, n \geq 1$, there is a total computable $(m+1)$-ary function $s_n^m(e, \bar{x})$ such that

$$\phi_e^{(m+n)}(\bar{x}, \bar{y}) \simeq \phi^{(n)}_{s_n^m(e,\bar{x})}(\bar{y}).$$

where $\bar{x}, \bar{y}$ is a tuple with length m and n respectively.

PROOF | 内容... ■

SECTION 4.5

Selected exercise of ch.4

Exercise 4.4.3 (p.84) Let $n \geq 1$. Show that there is a total computable function s such that

$$W_{s(x)}^{(n)} = \{(y_1, \dots, y_n) \mid y_1 + y_2 + \cdots + y_n = x\}.$$

Proof Define a $n+1$-ary function f by

$$f(x, y_1, \dots, y_n) = \begin{cases} 666 & x = y_1 + \cdots + y_n \\ \text{undefined} & \text{otherwise} \end{cases}$$ [10]

Clearly f is computable, then $f = \phi_e^{(n+1)}$ for some index e.

☞ But the indexes of f are not *unique*, hence we suppose that e is the smallest index, in other words, e is a constant here. (If we don't consider e as a constant, then we have another method to deal with that question in the following.)

By the **s-m-n Theorem** (general form), there is a total computable function $s_n^1(e, x)$ such that

$$\phi_{s_n^1(e,x)}^{(n)}(y_1, \dots, y_n) \simeq \phi_e^{(n+1)}(x, y_1, \dots, y_n) \simeq f(x, y_1, \dots, y_n).$$

Let $s(x) = s_n^1(e, x)$ (note that e is a constant here), then

$$\phi_{s(x)}^{(n)}(y_1, \dots, y_n) \simeq f(x, y_1, \dots, y_n).$$

Therefore by the construction of f, $W_{s(x)}^{(n)}$ is desired.

....

If we do not fix e, then the function $s_n^1(e, x)$ is a binary function indeed.
Using the **s-m-n Theorem** once more, let $s_n^1(e, x) = s_{k(e)}(x)$, since $s_n^1(e, x)$ is total, so is $s_{k(e)}(x)$. Then

$$W_{s_{k(e)}(x)}^{(n)} = \{(y_1, \dots, y_n) \mid y_1 + y_2 + \cdots + y_n = x\},$$

by our construction. [11] ■

[10] 此处的 *666* 可以是任何别的数

[11] 虚线下面的是袁老师的补充，但我还是感觉怪怪的，淦

CHAPTER 5

Universal programs

universal programs (UA): programs that in a sense embody all other programs.

some applications of UA: (to construct)

- non-computable functions
- undecidable predicates
- total computable function but not primitive recursive
- UA + s-m-n theorem

Key words: UA,

Section 5.1

Universal functions and universal programs

Let

$$\psi(x, y) \simeq \phi_x(y),$$

clearly, ψ embodies all the unary computable functions $\phi_0, \phi_1, \phi_2, \ldots$. Hence ψ is the **universal function** for unary computable functions.

Definition 5.1 (Universal function) The **universal function** for n-ary computable functions is the $(n+1)$ -ary function $\psi_U^{(n)}$ defined by

$$\psi_U^{(n)}(e, x_1, \ldots, x_n) \simeq \phi_e^{(n)}(x_1, \ldots, x_n).$$

We write ψ_U for $\psi_U^{(1)}$.

Is ψ_U computable?

If so, then any program P that computes ψ_U would appear to embody all other programs, hence P is called **universal program**.

A point is that a **universal program** P does not need to contain all instructions of all other program P_e in itself; P only needs the ability to *decode* any number e and hence *mimic* P_e.

Theorem 5.2 For each n, then universal function $\psi_U^{(n)}$ is computable.

Proof 内容… ■

Corollary 5.3 For each $n \geq 1$, the following predicates are decidable.

1. $S_n(e, \bar{x}, y, t) \equiv$ '$P_e(\bar{x})\downarrow y$ in t or fewer steps'.
2. $H_n(e, \bar{x}, t) \equiv$ '$P_e(\bar{x})\downarrow$ in t or fewer steps'.

$S_n(e, \bar{x}, y, t)$: 第 e 个程序 P，以 $\bar{x}$ 为输入，在小于等于 t 步内停机，且输出 y。
$H_n(e, \bar{x}, t)$: 第 e 个程序 P，以 $\bar{x}$ 为输入，在小于等于 t 步内停机。
n 是输入 $\bar{x}$ 的维度，即表示了 n 元函数。

Theorem 5.4 (Kleene's normal form theorem) There is a total computable function $U(x)$, and for each $n \geq 1$ a decidable predicate $T_n(e, \bar{x}, z)$ such that (n is the length of $\bar{x}$)

1. $\phi_e^{(n)}(\bar{x})$ is defined 当且仅当 $\exists z : T_n(e, \bar{x}, z)$.
2. $\phi_e^{(n)}(\bar{x}) \simeq U(\mu z\, T_n(e, \bar{x}, z))$.

Section 5.2

Effective operations on computable functions

Section 5.3

Selected Exercises

CHAPTER

6

Decidability, Undecidability & Partial Decidability

Table 6.1. Content of Chapter 6

A summary see Table 2

Target the limitations of computability.
theoretical limits
Sect. 1 discusses some methods for establishing undecidability.
Sect. 2-5 are devoted to a sample of decidable and undecidable problems from other areas of math. [skip]
Sect. 6: discussing *partial decidability*.

Keywords Diagonal construction, Reducing, Rice's Theorem,

...

Recap A n-ary predicate $M(\bar{x})$ is said to be **decidable** if its **characteristic function** c_M, given by

$$c_M(\bar{x}) = \begin{cases} 1 & \text{if } M(\bar{x}) \text{ hold} \\ 0 & \text{if } M(\bar{x}) \text{ doesn't hold} \end{cases}$$

is computable. (note that n is the length of sequence $\bar{x}$)

An algorithm for computing c_M is called a **decision procedure** for $M(\bar{x})$.

Section 6.1

Undecidable problems

Diagonal construction.

Theorem 6.1 (**An important Undecidable result**) The predicate '$x \in W_x$' is undecidable [12].

Equivalently, '$\underline{\phi_x(x) \text{ is defined}}$', '$\underline{P_x(x)\downarrow}$', '$\underline{\psi_U(x,x) \text{ is defined}}$'[13] is undecidable.

[12] *W_x is the domain of ϕ_x.*

[13] *note that the universal function ψ_U is* not total.

Proof .... method 1 (by Prof. Yuan) ....
ddd
.... method 2 (p.101 in Pink Book) ....

The characteristic function f of this problem is given by

$$f(x) = \begin{cases} 1 & x \in W_x \\ 0 & x \notin W_x. \end{cases}$$

It suffices to show that f is uncomputable.

Suppose for the sake that f is computable, we shall obtain a contradiction. Following we will make a **diagonal construction** of a function g such that

'f is computable implies g is computable'.

But in fact we will prove that g is not computable, hence f is not computable. Contradiction!
Define g by

$$g(x) = \begin{cases} 0 & x \notin W_x \text{ (i.e. } f(x) = 0) \\ undefined & x \in W_x \text{ (i.e. } f(x) = 1) \end{cases}$$

Obviously if f is computable then so is g.
But $Dom(g) \neq W_x = Dom(\phi_x)$, in detail, if g is computable, then $g = \phi_m$ for some index m, hence $m \in W_x \Leftrightarrow m \in Dom(g) \Leftrightarrow m \notin W_m$, contradiction.

We conclude that f is not computable, and so the problem '$x \in W_w$' is undecidable. ■

Remark the above theorem does *no say* that we cannot tell for any *particular* number a whether $\phi_a(a)$ is defined.

What the Theorem says is that, there is no single *general method* for deciding whether $\phi_x(x)$ is defined, i.e. there is no method that works for *every* x.

Corollary 6.2 There is a *computable* function h such that the problems '$x \in Dom(h)$' and '$x \in Ran(h)$' are both undecidable.

Proof Let

$$h(x) = \begin{cases} x & x \in W_x \\ \text{undefined} & x \notin W_x \end{cases}$$

More formally,

$$h(x) \simeq x \cdot \mathbf{1}(\psi_U(x,x))^{14}$$

[14] ***1*** *is 1-function such that* ***1***(y) *= 1.*

Clearly h is computable.
$x \in Dom(h) \Leftrightarrow x \in Ran(h) \Leftrightarrow x \in W_x$, but '$x \in W_x$' is undecidable. ■

Another important undecidable problem:

Theorem 6.3 (**the Unsolvability of the Halting Problem**) The problem '$\phi_x(y)$ is defined' is undecidable.

Equivalently, '$P_x(y)\downarrow$' or '$y \in W_x$' is undecidable.

Proof If '$\phi_x(y)$ is defined' is decidable, then so is the problem '$\phi_x(x)$ is defined', which contradicts with Theorem 6.1. ■

The **Halting Problem** says that: there is no effective general method for discovering whether a given program running on a given input eventually halts.

Remark (**Reducing method**) The problem '$x \in W_x$' is important for several reasons. Many problems can be shown to be undecidable by showing that they are at least as difficult as this one, for example, the **Halting Problem**.

This process is known as **reducing** one problem to another.

Often we can show that a solution to the general problem $M(x)$ would lead to a solution to the problem '$x \in W_x$'. Then we say '$x \in W_x$' is **reduced** to $M(x)$.

Notation $x \in W_x \preceq M(x)$

That is, the decidability of $M(x)$ implies the decidability of '$x \in W_x$', from which we conclude immediately that $M(x)$ is undecidable.

Theorem 6.4 The problem '$\phi_x = \mathbf{0}$' is undecidable.

PROOF Using Rice's Theorem here. [15]

Let $R = \{\mathbf{0}\}$, clearly $\emptyset \neq R \subset \mathscr{C}_1$. Then $\phi_x = \mathbf{0} \Leftrightarrow \phi_x \in R$, by Rice's Theorem, the problem '$\phi_x = \mathbf{0}$' is undecidable. ■

[15] *a more complex proof can be find in p.103 of Pink Book .*

The above theorem shows that there can be no perfectly general effective method for checking whether a program will computer the zero function. In fact, we can see that the same is true for any particular computable function.

The following corollary shows that the question of whether two programs compute the same unary function id undecidable.

Corollary 6.5 The problem '$\phi_x = \phi_y$' is undecidable.

PROOF Let c be a index such that $\phi_c = \mathbf{0}$. Then '$\phi_x = \phi_y$' is undecidable implies '$\phi_x = \phi_c$' is undecidable, that contradicts with Theorem 6.4. ■

Theorem 6.6 (**Input and Output Problem**) Let c be any number, the following problems are undecidable:

1. Input Problem: '$c \in W_x$' , equivalently, '$P_x(c)\downarrow$' or '$c \in Dom(\phi_x)$'.
2. Output Problem: '$c \in E_x$' [16] , equivalently, $c \in Ran(\phi_x)$.

[16] *E_x is the range of ϕ_x.*

PROOF Using Rice's Theorem here.
(a) Let $R = \{g \in \mathscr{C}_1 \mid c \in Dom(g)\}$. R is non-empty since it contains all total unary computable functions, and $R \neq \mathscr{C}_1$ obviously.

Then $c \in Dom(\phi_x) \Leftrightarrow \phi_x \in R$, by Rice's Theorem, the undecidable result is desired.
(b) Let $R = \{g \in \mathscr{C}_1 \mid c \in Ran(g)\}$. R is non-empty since it must contains **c**-functions, and $R \neq \mathscr{C}_1$ obviously.

Then $c \in Rom(\phi_x) \Leftrightarrow \phi_x \in R$, by Rice's Theorem, the undecidable result is desired.

. . . .

A more complex simultaneously proof can be find in p.105 of Pink Book. But note that, for $f(x, y)$ given by

$$f(x,y) = \begin{cases} y & x \in W_x \\ undefined & x \notin W_x \end{cases}$$

if we exchange y with c, then it is also working. ■

Following is a very general undecidability result.

Theorem 6.7 (**Rice's Theorem**) Suppose that $R \subseteq \mathscr{C}_1$, and $R \neq \emptyset, \mathscr{C}_1$ [17]. Then the problem '$\phi_x \in R$' is undecidable.

[17] *recall that $\mathscr{C}_1$ is the class of unary computable functions. This is, R is a non-empty proper subclass of $\mathscr{C}_1$.*

PROOF By the **Algebra of Decidability** (Theorem **??**) , we know that

'$\phi_x \in R$' is decidable $\Leftrightarrow$ '$\phi_x \in \mathscr{C}_x \setminus R$' is decidable. [18]

[18] *p.37 in Pink Book*

■

Thinking ? 6.8 任给一个模态框架类 F 和 $\mathfrak{F}$，$\mathfrak{F} \in \mathcal{F}$ 是不可判定的吗?

SECTION 6.2

Mathematical Logic

没怎么搞懂，要多看几遍

Decidability in logic:

Propositional calculus is decidable: use *truth table*.

Provability and **Validity** in predicate calculus is *undecidable*. [Church, 1936]

This is the most fundamental undecidability result for the whole of mathematics. [Hilbert]

Following we use URM to give an easy proof of the *undecidability of validity*.

Theorem 6.9 (the Undecidability of FOL) Validity in the first-order predicate calculus is undecidable.

PROOF Let P be a program in *standard form* having instructions $I_1, \ldots, I_s$.
Let $u = \rho(P)$, that is, the smallest number of registers affected by P.
We use following symbols of the predicate calculus:

0	a symbol for an individual
'	a symbol for unary function (whose value at x is x')
R	a symbol for a $(u+1)$-ary relation
$x_1, x_2, \ldots, x_u, y$	symbols for variable individuals

The interpretation of these symbols is that

- 0 represents the number 0
- ' represents the successor function $x + 1$, we write 1 for $0'$, 2 for $0''$, etc.
- R represents the possible states of a computation under P.

  $\mathsf{R}(r_1, \ldots, r_u, k)$ where $r_1, \ldots, r_u, k \in \mathbb{N}$ means that the state

r_1	r_2	$\cdots$	r_u	0	0	$\cdots$

next instruction I_k

occurs in the computation.

Now for each instruction I_i ($1 \leq i \leq s$), we can write down a statement τ_i of the predicate calculus that describes the effect of I_i on states, using the boolean symbol $\wedge, \rightarrow$:

1. if $I_i = Z(n)$ (note that it is must be $n \leq u$), let τ_i be
   $\forall x_1 \ldots \forall x_u : R(x_1, \ldots, x_n, \ldots, x_u, i) \rightarrow R(x_1, \ldots, 0, \ldots, x_u, i')$
2. if $I_i = S(n)$, let τ_i be
   $\forall x_1 \ldots \forall x_u : R(x_1, \ldots, x_n, \ldots, x_u, i) \rightarrow R(x_1, \ldots, x_n', \ldots, x_u, i')$
3. if $I_i = T(m, n)$ (note that it is must be $m, n \leq u$), let τ_i be
   $\forall x_1 \ldots \forall x_u : R(x_1, \ldots, x_n, \ldots, x_u, i) \rightarrow R(x_1, \ldots, x_m, \ldots, x_u, i')$
4. if $I_i = J(m, n, q)$, let τ_i be
   $\forall x_1 \ldots \forall x_u : R(x_1, \ldots, x_u, i) \rightarrow$
   $(x_m = x_n \rightarrow R(x_1, \ldots, x_u, q)) \wedge \ (x_m \neq x_n \rightarrow R(x_1, \ldots, x_u, i'))$

Let

$$\tau_0 := \forall x \forall y((x = y \rightarrow x' = y') \wedge x' \not 0)$$

Now for any $a \in \mathbb{N}$ let σ_a be the statement

$$\sigma_a := (\tau_0 \wedge \tau_1 \wedge \cdots \tau_s \wedge R(a, 0, \ldots, 0, 1)) \rightarrow \exists x_1 \ldots \exists x_u R(x_1, \ldots, x_u, s+1)$$

$R(a, 0, \ldots, 0, 1)$ corresponds to a *staring state*

a	0	0	0	0	0	$\ldots$

next instruction I_1

and any statement $R(x_1, \ldots, x_u, s+1)$ corresponds to a *halting state* since there is no instruction I_{s+1}.
Thus we shall see that

$$(*) \qquad P(a)\downarrow \Leftrightarrow \ \sigma_a \text{ is valid } .$$

$\Rightarrow$
Suppose $P(a)\downarrow$and we have a structure in which $\tau_0 \wedge \tau_1 \wedge \cdots \tau_s \wedge R(a, 0, \ldots, 0, 1)$ hold. Using the statements $\tau_0, \ldots, \tau_s$ we find that each of the statements $R(r_1, \ldots, r_u, k)$ corresponding to the successive states in the computation also holds.
Eventually we find that a halting statement $R(b_1, \ldots, b_u, s+1)$ holds for some $b_1, \ldots, b_u \in \mathbb{N}$, and hence $\exists x_1 \ldots \exists x_u R(x_1, \ldots, x_u, s+1)$ holds.
Thus σ_a is valid.

$\Leftarrow$
If σ_a is valid, it holds in particular in the structure $\mathbb{N}$ with the predicate symbol R interpreted by the predicate R_a where
$R_a(a_1, \ldots, a_u, k) \equiv$ At some stage in the computation $P(a)$ the registers contains $a_1, a_2, \ldots, a_u, 0, 0, \ldots$ and the next instruction is I_k.
Then $\tau_0, \ldots, \tau_s$ and $R(a, 0, \ldots, 0, 1)$ all hold in this structure, hence so does $\exists x_1 \ldots \exists x_u R(x_1, \ldots, x_u, s+1)$. Therefore $P(a)\downarrow$.

If we take P to be a program that computes the function $\psi_U(x, x)$, the $(*)$ gives a reduction of the problem '$x \in W_x$' to the problem 'σ is valid'. Hence the latter is undecidable. ■

. . . .
'σ is true in the field of real numbers' is decidable [Tarski, 1951]

Section 6.3
Partially decidable predicates

CHAPTER

7

Recursive and recursively enumerable sets

Table 7.1. Contents of Ch.7

Target | 内容...

Keywords | 内容...

Section 7.1
Recursive sets

Section 7.2
Recursively enumerable sets

Section 7.3
Productive and creative sets

Section 7.4
Simple sets

Section 7.5
Selected exercises

CHAPTER

8

Incompleteness

Section 8.1

rograms

Section 8.2

kkk

Appendix I: some computable functions in the pink book

汇总 pink book 中出现的一些常见的可计算函数

CHAPTER

9

testing

Section 9.1

amsthm Environments

amsthm environments are defined as usual being enclosed by \begin{environment}··· \end{environment}. Modifications include integration with the tcolorbox package.

Note that counting for theorems and lemmas is distinct from the counting for definitions. Also, the breakable option for tcolorbox allows these environments to span multiple pages.

If one wishes to change the color, simply modify the line which states borderline west={1pt}{0pt}{blue}. The first numeric value dictates the width of the line, the second dictates how close it is away from the *left* margin, while the last argument declares the color. This customization is independent of the amsthm environments.

There is one issue with this however. Since we are using a tcolorbox, this proof environment is incompatible with \sn and \sidenote, as it results in a **Float(s) Error**. However, this environment is compatible with \mn and \marginnote.

中文测试爱你的 and this is English

Definition 9.1 (**Modal and Frame**) The definition environment and the associated tcolorbox are provided by the following code in NotesTeX.sty:

```
\tcolorboxenvironment{definition}{
  boxrule=0pt,
  boxsep=0pt,
  colback={White!90!Cerulean},
  enhanced jigsaw,
  borderline west={2pt}{0pt}{Cerulean},
  sharp corners,
  before skip=10pt,

  after skip=10pt,
  breakable,
}
```

dsfasdfasdfadf adsfa sdf sdfa dsf fasd asdf asdf asdf asdf asdf

Theorem 1 The theorem environment and the associated tcolorbox are provided by the following code in NotesTeX.sty:

```
\tcolorboxenvironment{theorem}{
  boxrule=0pt,
  boxsep=0pt,
  colback={White!90!Dandelion},
```

```
  enhanced jigsaw,
  borderline west={2pt}{0pt}{Dandelion},
  sharp corners,
  before skip=10pt,
  after skip=10pt,
  breakable,
}
```

Lemma 9.2 The `lemma` environment and the associated `tcolorbox` are provided by the following code in `NotesTeX.sty`:

```
\tcolorboxenvironment{lemma}{
  boxrule=0pt,
  boxsep=0pt,
  blanker,
  borderline west={2pt}{0pt}{Red},
  before skip=10pt,
  after skip=10pt,
  sharp corners,
  left=12pt,
  right=12pt,
  breakable,
}
```

ddd

Prop. 9.3 sdfasdfas

dddd
ddddd

PROOF The `proof` environment and the associated `tcolorbox` are provided by the following code in `NotesTeX.sty`:

```
\tcolorboxenvironment{proof}{
  boxrule=0pt,
  boxsep=0pt,
  blanker,
  borderline west={2pt}{0pt}{NavyBlue!80!white},
  before skip=10pt,
  after skip=10pt,
  left=12pt,
  right=12pt,
  breakable,
}
```

■

Example 9.4 The `example` environment and the associated `tcolorbox` are provided by the following code in `NotesTeX.sty`:

```
\tcolorboxenvironment{example}{
  boxrule=0pt,
  boxsep=0pt,
  blanker,
  borderline west={2pt}{0pt}{Black},
  sharp corners,
```

```
  before skip=10pt,
  after skip=10pt,
  left=12pt,
  right=12pt,
  breakable,
}
```

Remark The `remark` environment and the associated `tcolorbox` are provided by the following code in `NotesTeX.sty`: [19]

```
\tcolorboxenvironment{remark}{
  boxrule=0pt,
  boxsep=0pt,
  blanker,
  borderline west={2pt}{0pt}{Green},
  before skip=10pt,
  after skip=10pt,
  left=12pt,
  right=12pt,
  breakable,
}
```

[20]

[19] *Coexistence of `amsthm` environment and `mn`*

[20] *ddddd of `amsthm` environment and `mn`*

Section 9.2

Fullpage Environment

The `fullpage` environment is defined by

```
\begin{fullpage}
...
\end{fullpage}
```

with the width of the `fullpage` environment given by `\textwidth`+`\marginparsep`+`\marginparwidth`.The code in `NotesTeX.sty` that is responsible for the `fullpage` environment is given by

```
\newenvironment{fullpage}{
{\smallskip\noindent
\begin{minipage}{\textwidth+\marginparwidth+\marginparsep}\hrule\smallskip\smallskip}
{\smallskip\smallskip\hrule\end{minipage}\vspace{.1in}
}
```

....

Remark Eliminating the `\hrule` in the code will remove the lines surrounding the `fullpage` environment. Similarly, it is possible to change the vertical spacing after the `fullpage` is over, by modifying the `\vspace{}` argument.

---

lec entry

`multicols` may be used in conjunction with `fullpage`. I find it useful for formatting exercises in multiple columns and it makes the text distinct from the rest of the `fullpage` environment. The `lec` environment is compatible with `multicols` but `sidenote`, `marginnote` are not.

Subsection 9.2.1

Known Issues with Fullpage

Remark Since the `fullpage` environment uses a `minipage`, and minipages do not work over multiple pages, one will need a new `fullpage` per page.

Remark If the `twoside` option is enabled in the `documentclass` header, then the `fullpage` is known to bleed out beyond the margin.

this is a subsubsection

dsdfa sdf
asdf

Section 9.3

van Benthem Characterization Theorem

today we are going to prove the famous van Benthem Characterization Theorem.

Languages:

the basic modal language and its first-order language

$$\mathcal{L}_{ml} \ni \varphi ::= p \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \Diamond\varphi.$$
$$\mathcal{L}^1 \ni \varphi ::= x \equiv x \mid P_i x \mid Rxy \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \forall x\varphi.$$

Definition 9.5 (**Frames and Models**) Frame: Model:

Theorem 2 内容...

Theorem 3 (thsdf) 内容... dsdf

[1] ddd sd

box content

ddd dsds

Definition 9.6 this is a definition

Theorem 9.7 (asdfa) this is a theorem 这是一条定理

Proof (sdfsdfa) 内容... dsf sdf adsfasdf ■

Corollary 9.8 (Corollary) 内容... this is a corollary.

Example 9.9 (Modal and Frame) 内容... dsfasdf sadfhiasdofh iuasdhfkhak jhkajdskfjh as fasdf asdfa adsf asdfa [21] asdf f sdf a

[21] *this is a sidenote hhhhh*

this is a marginnote dddd

Lemma 9.10 (Lindenbaum Lemma) 内容...

Prop. 9.11 this is a proposition.

Example 9.12 this is a example

Fact 9.13 this is a fact

claim 风格的环境:
内容...
内容...
内容...

Remark 内容...

Bibliography

[1] T. Bolander and P. Blackburn. Termination for hybrid tableaus. *Journal of Logic and Computation*, 17(3):517–554, March 2007.