Notes on Computability Theory

Stefan Vatev

January 16, 2023

Contents

Chapter 1

Primitive recursive functions

1.1 Definition and examples

- The following are called **basic functions** over the natural numbers:

The main reference for this chapter is [7] and [2].

$$\begin{aligned} &\mathtt{S}(n) = n+1 && \text{// the successor function} \\ &\mathtt{O}(n) = 0 && \text{// the zero function} \\ &\mathtt{I}^n_k(x_1,\ldots,x_k,\ldots,x_n) = x_k, && \text{// the projective functions,} \\ & && \text{// for all } n \text{ and } k \le n \end{aligned}$$

- Let $g : \mathbb{N}^n \to \mathbb{N}$, $f_1 : \mathbb{N}^k \to \mathbb{N}, \ldots, f_n : \mathbb{N}^k \to \mathbb{N}$ are functions. Define the function $h : \mathbb{N}^k \to \mathbb{N}$ as

Here all functions are total, but we can define superposition for partial functions too.

$$h(\bar{x}) = z \;\Leftrightarrow\; (\exists y_1 \cdots \exists y_n)[f_1(\bar{x}) = y_1 \,\&\ldots\&\, f_n(\bar{x}) = y_n \,\&\, g(y_1,\ldots,y_n) = z].$$

  We say that h is the **superposition** of g with $f_1, \ldots, f_n$. We denote

$$h \overset{\text{def}}{=} g(f_1,\ldots,f_n).$$

  If we work only with total functions, it is safe to write that

$$h(\bar{x}) = g(f_1(\bar{x}),\ldots,f_n(\bar{x})).$$

  If we allow partial functions, we must be more careful, because we may have that $f_i(\bar{x})$ is undefined, for some i. It follows that in this case $h(\bar{x})$ is also undefined.

- We say that h is obtained from f and g by **primitive recursion** if the definition of h follows the scheme:

$$\left| \begin{array}{lcl} h(\overline{x},0) & = & f(\overline{x}) \\ h(\overline{x},n+1) & = & g(\overline{x},n,h(\overline{x},n)) \end{array} \right.$$

- We say that one function is **primitive recursive** if it can be produced from the basic functions by applying a finite number of times the operations of *superposition* and *primitive recursion*.

Having a formal definition will be useful when we want to prove a property which holds for all primitive recursive functions.

Primitive recursion was first introduced explicitly by Thoralf Skolem in 1923. Rózsa Petér [7] introduced the term "primitive recursive".

Definition 1.1 (Primitive recursion)**.** The primitive recursive functions are formed by the rules:

1) The basic functions are primitive recursive;
2) If $g : \mathbb{N}^n \to \mathbb{N}$ and $f_1 : \mathbb{N}^k \to \mathbb{N}, \ldots, f_n : \mathbb{N}^k \to \mathbb{N}$ are primitive recursive, then the function $g(f_1, \ldots, f_n) : \mathbb{N}^k \to \mathbb{N}$ is also primitive recursive.
3) If $f : \mathbb{N}^n \to \mathbb{N}$ and $g : \mathbb{N}^{n+2} \to \mathbb{N}$ are primitive recursive, then $h : \mathbb{N}^{n+1} \to \mathbb{N}$, obtained from f and g by the primitive recursion scheme, is also primitive recursive.
4) All primitive recursive functions are produced by the rules 1) - 3).

Theorem 1.1. All primitive recursive functions are total.

Hint. Straightforward induction on the definition of the primitive recursive functions. □

Examples of primitive recursive functions

Here we list a number of useful primitive recursive functions.

1) We will show that if f is primitive recursive, then h with the property $h(x, y) = f(y, x)$ is also primitive recursive. Consider the function

It is easy to generalise it to n-ary functions.

$$h(x, y) \overset{\text{def}}{=} f(\mathtt{I}^2_2, \mathtt{I}^2_1)(x, y).$$

   - By definition, $\mathtt{I}^2_1$ and $\mathtt{I}^2_2$ are primitive recursive.
   - By assumption, f is primitive recursive.
   - h is obtained from $\mathtt{I}^2_1$, $\mathtt{I}^2_2$ and f by superposition, i.e. $h = f(\mathtt{I}^2_2, \mathtt{I}^2_1)$.
   - It follows that the function h is primitive recursive.

2) Let $\texttt{plus}(x, y) = x + y$. We can define the function $\texttt{plus}$ in the following way:

From here on, we will use $+$ instead of $\texttt{plus}$.

$$\left| \begin{array}{lll} \texttt{plus}(x, 0) & \overset{\text{def}}{=} & x = \texttt{I}_1^1(x) \\ \texttt{plus}(x, y+1) & \overset{\text{def}}{=} & \texttt{plus}(x, y) + 1 = \texttt{S}(\texttt{plus}(x, y)) = g(x, y, \texttt{plus}(x, y)), \end{array} \right.$$

where $g(x, y, z) \overset{\text{def}}{=} \texttt{S}(\texttt{I}_3^3)(x, y, z) = \texttt{S}(z)$. It follows that $\texttt{plus}$ is primitive recursive.

3) Let $\texttt{mult}(x, y) = x \cdot y$. We can define $\texttt{mult}$ in the following way:

Of course, later, we will use $*$ instead of $\texttt{mult}$.

$$\left| \begin{array}{lll} \texttt{mult}(x, 0) & \overset{\text{def}}{=} & 0 = \texttt{O}(x) \\ \texttt{mult}(x, y+1) & \overset{\text{def}}{=} & \texttt{mult}(x, y) + x = g(x, y, \texttt{mult}(x, y)), \end{array} \right.$$

where the function g is defined as

$$g(x, y, z) \overset{\text{def}}{=} \texttt{plus}(\texttt{I}_3^3, \texttt{I}_1^3)(x, y, z).$$

Clearly g satisfies the property that $g(x, y, z) = z + y$.

- By definition, $\texttt{I}_1^3$ and $\texttt{I}_3^3$ are primitive recursive.
- We have already seen that $\texttt{plus}$ is primitive recursive.
- F is obtained from $\texttt{I}_1^3$, $\texttt{I}_3^3$ and $\texttt{plus}$ by superposition.
- It follows that g is primitive recursive.
- Since $\texttt{mult}$ is obtained from $\texttt{O}$ and g by the primitive recursive scheme, it follows that $\texttt{mult}$ is primitive recursive.

4) Given a unary function f, we let $f^\star(x, n) \overset{\text{def}}{=} f^{(n)}(x)$, where we use

✍ **Do it!**

$$\left| \begin{array}{lll} f^{(0)}(x) & \overset{\text{def}}{=} & x \\ f^{(n+1)}(x) & \overset{\text{def}}{=} & f(f^{(n)}(x)). \end{array} \right.$$

Show that if f is primitive recursive, then $f^\star$ is primitive recursive.

Now we may let $\texttt{plus}(x, y) = S^\star(x, y)$.

5) Let $h(x, y) = \prod_{i<y} f(x, i)$. We will show that if f is primitive recursive, then h is primitive recursive. We can define h in the following way:

This is called bounded product. Notice that $\prod_{i<0} i = 1$.

$$\left| \begin{array}{lll} h(x, 0) & = & 1 = \texttt{S}(\texttt{O}(x)) = g_0(x) \\ h(x, n+1) & = & h(x, n) \cdot f(x, n) = g(x, n, h(x, n)), \end{array} \right.$$

where the function g is defined as

$$g(x, y, z) \overset{\text{def}}{=} \texttt{mult}(z, f(\texttt{I}_1^3(x, y, z), \texttt{I}_2^3(x, y, z))).$$

- g_0 is obtained from $\mathtt{O}$ and $\mathtt{S}$ by superposition and hence g_0 is primitive recursive.
- It is easy to see that g is primitive recursive, because it is a superposition of primitive recursive functions.
- Finally, h is obtained from g_0 and g by following a primitive recursive scheme. It follows that h is primitive recursive.

6) Similarly, we can show that if f is primitive recursive, then

$$h(x,y) = \sum_{i<y} f(x,i)$$

is primitive recursive.

This is called bounded summation. Notice that $\sum_{i<0} i = 0$.

7) We will show that the following function is primitive recursive:

This function is very useful. Remember it!

$$\mathtt{sign}(x) = \begin{cases} 0, & \text{if } x = 0 \\ 1, & \text{if } x > 0. \end{cases}$$

Consider the following primitive recursive scheme:

$$\left| \begin{array}{lcl} f(x,0) & = & \mathtt{O}(x) \\ f(x,y+1) & = & \mathtt{S}(\mathtt{O}(x)) = h(x,y,f(z,y)), \end{array} \right.$$

where $h(x,y,z) \overset{\text{def}}{=} \mathtt{S}(\mathtt{O}(\mathtt{I}^3_1(x,y,z))$.

- h is primitive recursive as a superposition of primitive recursive functions.
- f is obtained from O and H following the primitive recursive scheme. Thus, f is primitive recursive.
- Then $\mathtt{sign}(x) = f(x,x) = f(\mathtt{I}^1_1, \mathtt{I}^1_1)(x)$ is a superposition of primitive recursive functions and hence $\mathtt{sign}$ is primitive recursive.

8) Similarly, we can show that the function

✍ **Do it!**

$$\overline{\mathtt{sign}}(x) = \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{if } x > 0 \end{cases}$$

is primitive recursive.

9) Show that if f, g, and p are primitive recursive, then h is primitive recursive, where:

$$h(x) = \begin{cases} f(x), & \text{if } p(x) = 0, \\ g(x), & \text{if } p(x) \neq 0. \end{cases}$$

Since we have that `sign` is primitive recursive, we can give the following definition of h:

Notice that this definition of h works only for *total* functions.

$$h(x) \overset{\text{def}}{=} \overline{\texttt{sign}}(p(x)) \cdot f(x) + \texttt{sign}(p(x)) \cdot g(x).$$

10) Our next step is to show that the predecessor function

$$\texttt{pred}(x) = \begin{cases} 0, & \text{if } x = 0 \\ x - 1, & \text{if } x \geq 1 \end{cases}$$

is primitive recursive. Consider the following function:

$$\left| \begin{array}{lcl} \texttt{pred}'(x, 0) & = & \texttt{0}(x) \\ \texttt{pred}'(x, n+1) & = & \texttt{I}_2^3(x, n, \texttt{pred}'(x, n)), \end{array} \right.$$

- $\texttt{pred}'$ is obtained from $\texttt{0}$ and $\texttt{I}_2^3$ following the primitive recursive scheme and thus it is primitive recursive. It is easy to see that $\texttt{pred}(x) = \texttt{minus}'(x, x)$. Thus, the function `pred` is primitive recursive.

The first argument of $\texttt{pred}'$ is not used. It is there just to fit in the definition into the primitive recursive scheme.

11) We will show that the following function is primitive recursive:

Some people call $\dot{-}$ modified minus, or monus.

$$x \dot{-} y = \begin{cases} 0, & \text{if } x < y \\ x - y, & \text{if } x \geq y \end{cases}.$$

In particular, $x \dot{-} 1 = \texttt{pred}(x)$. This function turns out to be very useful, so we will do this carefully. We will show that $x \dot{-} y$ can be obtained by the primitive recursion scheme by using the following property:

$$x \dot{-} (y + 1) = (x \dot{-} y) \dot{-} 1.$$

Now it is east to see that $x \dot{-} y$ is primitive recursive, because it has the following definition:

$$\left| \begin{array}{lcl} \texttt{minus}(x, 0) & \overset{\text{def}}{=} & x \\ \texttt{minus}(x, n+1) & \overset{\text{def}}{=} & \texttt{pred}(\texttt{minus}(x, n)). \end{array} \right.$$

Alternatively, we can show that `minus` is primitive recursive by using 4) and observing that

$$\texttt{minus}(x,y) = \texttt{pred}^{\star}(x,y),$$

i.e. $x \dot{-} y = (\cdots(x \dot{-} \underbrace{1) \dot{-} 1) \cdots \dot{-} 1}_{y \text{ times}}$.

12) Now it is easy to see that the functions $\min(x,y)$ and $\max(x,y)$ are primitive recursive. One way to do it is the following:

✍ Give primitive recursive definitions of $\min(x,y,z)$ and $\max(x,y,z)$.

$$\min(x,y) = x \dot{-} (x \dot{-} y)$$
$$\max(x,y) = x + (y \dot{-} x).$$

13) The function $|x-y|$ is primitive recursive, because we can define it in the following way:

$$|x-y| = (x \dot{-} y) + (y \dot{-} x),$$

or like that:

$$|x-y| = \max(x,y) - \min(x,y).$$

Problem 1. Let $f(\bar{x},y)$ be a primitive recursive function. Show that the following functions are primitive recursive:

1) $h(\bar{x},y_1,y_2) = \begin{cases} \sum_{z=y_1}^{y_2} f(\bar{x},z), & \text{if } y_1 \leq y_2 \\ 0, & \text{if } y_1 > y_2. \end{cases}$

2) $h(\bar{x},y_1,y_2) = \begin{cases} \prod_{z=y_1}^{y_2} f(\bar{x},z), & \text{if } y_1 \leq y_2 \\ 1, & \text{if } y_1 > y_2. \end{cases}$

Problem 2. Let $f(\bar{x},y)$ and $g(\bar{x},y)$ be primitive recursive functions. Show that the following functions are primitive recursive:

1) $h(\bar{x},y) = \sum_{z<g(\bar{x},y)} f(\bar{x},z)$;

2) $h(\bar{x},y) = \prod_{z<g(\bar{x},y)} f(\bar{x},z)$;

1.2 Predicates

- A **predicate** is a *total* function $p : \mathbb{N}^n \to \{0, 1\}$, where we interpret 0 as *false* and 1 as *true*.
- To simplify matters, since we only work with natural numbers, let us introduce the constants $\mathtt{True} \overset{\text{def}}{=} 1$ and $\mathtt{False} \overset{\text{def}}{=} 0$.
- For an n-ary relation R on $\mathbb{N}$, its **characteristic function** is the function χ_R, where

$$\chi_R(\bar{x}) = \begin{cases} \mathtt{True}, & \text{if } \bar{x} \in R \\ \mathtt{False}, & \text{if } \bar{x} \notin R. \end{cases}$$

- We say that the relation R is **primitive recursive** if its characteristic function χ_R is primitive recursive.

1) Let us introduce the constant functions $\mathtt{true}(x) \overset{\text{def}}{=} \mathtt{True}$ and $\mathtt{false}(x) \overset{\text{def}}{=} \mathtt{False}$ for all x.

2) Let us first show that the following relation is primitive recursive:

$$\chi_<(x, y) = \begin{cases} \mathtt{True}, & \text{if } x < y \\ \mathtt{False}, & \text{if } x \geq y. \end{cases}$$

This is easy, because

$$\chi_<(x, y) = \mathtt{sign}(y \dot{-} x).$$

3) Similarly, $\chi_=(x, y)$ can be defined as

$$\chi_=(x, y) = \overline{\mathtt{sign}}(|x - y|).$$

4) Let P and Q are primitive recursive n-ary relations. Then $P \wedge Q$ is a primitive recursive relation because

$$\chi_{P \wedge Q}(\bar{x}) = \chi_P(\bar{x}) * \chi_Q(\bar{x}).$$

Similarly,

$$\chi_{P \vee Q}(\bar{x}) = \mathtt{sign}(\chi_P(\bar{x}) + \chi_Q(\bar{x})),$$

and

$$\chi_{\neg P}(\bar{x}) = 1 \dot{-} \chi_P(\bar{x}).$$

5) If R is a primitive recursive $(n+1)$-ary relation, then

$$Q(\bar{x}, y) \overset{\text{def}}{=} (\exists z < y) R(\bar{x}, z)$$

is a primitive recursive relation, because

$$\chi_Q(\bar{x}, y) = \texttt{sign}(\sum_{z<y} \chi_R(\bar{x}, z)).$$

Of course, here it is essential that we defined $\texttt{True} = 1$ and $\texttt{False} = 0$.

6) Similarly, if R is a primitive recursive $(n+1)$-ary relation, then

$$Q(\bar{x}, y) \overset{\text{def}}{=} (\forall z < y)[R(\bar{x}, z)]$$

is a primitive recursive relation, because

$$\chi_Q(\bar{x}, y) = \prod_{z<y} \chi_R(\bar{x}, z).$$

1.3 Bounded minimisation

Let f be a $(k+1)$-ary *total* function. We say that the function g is obtained from f by **bounded minimisation** if

$$g(\bar{x}, y) = \begin{cases} \min\{z \mid z < y \;\&\; f(\bar{x}, z) = 0\}, & \text{if } (\exists z < y)[f(\bar{x}, z) = 0] \\ y, & \text{otherwise.} \end{cases}$$

We usually denote this in the following way:

$$g(\bar{x}, y) = (\mu z < y)[f(\bar{x}, z) = 0].$$

Theorem 1.2. If f is primitive recursive function and g is obtained from f by **bounded minimisation**, then g is also primitive recursive. In other words, the class of primitive recursive functions is closed under bounded minimisation.

Hint. We can define g in the following way:

$$g(\bar{x}, y) \overset{\text{def}}{=} \sum_{v=1}^{y} \texttt{sign}(\prod_{z=0}^{v \dot{-} 1} f(\bar{x}, z)).$$

□

Problem 3. Show that if $f(\bar{x}, y)$ is primitive recursive, then the following function is primitive recursive:

$$h(\bar{x}, y) = \begin{cases} \max\{z \mid z < y \;\&\; f(\bar{x}, z) = 0\}, & \text{if } (\exists z < y)[f(\bar{x}, z) = 0] \\ y, & \text{otherwise.} \end{cases}$$

Hint. We can define h in the following way:

$$h(\bar{x}, y) = y \dot{-} \sum_{i=1}^{y} \texttt{sign}(\prod_{z=y \dot{-} i}^{y \dot{-} 1} f(\bar{x}, z)).$$

□

We continue with examples of primitive recursive functions.

1) The function $\texttt{qt}(x, y)$ that gives the quotient of the division of y by x is primitive recursive. More formally, for $x > 0$, $\texttt{qt}(x, y) \stackrel{\text{def}}{=} q$, where $y = q*x+r$ for some r such that $0 \leq r < x$. Of course, we let $\texttt{qt}(0, y) \stackrel{\text{def}}{=} 0$. Then
$$\texttt{qt}(x, y) = \texttt{sign}(x) * (\mu z < y)[(z + 1) * x > y].$$

2) The function $\texttt{rem}(x, y)$ that gives the remainder of the division of y by x is primitive recursive. More formally, $\texttt{rem}(x, y) \stackrel{\text{def}}{=} r$, where r is the *least* natural number for which there exists a natural number q such that $y = q * x + r$. Since we know that $\texttt{qt}(x, y)$ is primitive recursive, then
$$\texttt{rem}(x, y) = y \dot{-} \texttt{qt}(x, y) * x.$$

3) The following function
$$\texttt{Div}(x, y) = \begin{cases} 1, & \text{if } x \mid y \\ 0, & \text{otherwise} \end{cases}$$
is primitive recursive since it can be defined as
$$\texttt{Div}(x, y) \stackrel{\text{def}}{=} \overline{\texttt{sign}}(\texttt{rem}(x, y)).$$

4) Now consider the function
$$\texttt{D}(x) = \begin{cases} \text{the number of divisors of } x, & \text{if } x > 0 \\ 0, & \text{if } x = 0. \end{cases}$$
It is easy to see that we can the function D in the following way:

Notice that $\texttt{D}(0) = 0 = \sum_{y=1}^{0}$

$$\texttt{D}(x) \stackrel{\text{def}}{=} \sum_{y=1}^{x} \texttt{Div}(y, x).$$

5) Let us consider the predicate
$$\texttt{Pr}(x) = \begin{cases} 1, & \text{if } x \text{ is a prime number} \\ 0, & \text{otherwise ;} \end{cases}$$
There are many ways to characterize the prime numbers.

- One way to characterize the prime numbers is by saying that the number of divisors is exactly 2. Thus,

$$\mathtt{Pr}(x) \overset{\text{def}}{=} \overline{\mathtt{sign}}(|\mathtt{D}(x) - 2|).$$

- Another way is by using the fact that n is prime iff $n \geq 2$ and there are no two numbers less that n whose product is n, i.e. n does not divide $(n-1)!^2$. Thus,

$$\mathtt{Pr}(x) \overset{\text{def}}{=} \overline{\mathtt{sign}}((2 \dot{-} x) + \overline{\mathtt{sign}}(\mathtt{rem}(n, (n-1)!^2))).$$

- Another way to do the same thing is by defining

$$\mathtt{Pr}(x) \overset{\text{def}}{=} \overline{\mathtt{sign}}(\prod_{a=2}^{n \dot{-} 1} \prod_{b=2}^{n \dot{-} 1} |a \cdot b - x|).$$

6) Consider the function $\mathtt{p}(n) = n$-th prime number, where $\mathtt{p}(0) = 2$. Our idea is to show that the function $\mathtt{p}(n)$ can be expressed by *bounded minimisation.* To do this, we need to find an upper bound for $\mathtt{p}(n)$ which depends on n.

   - It is easy to see that
   $$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1.$$

   - We will prove by induction on n that $\mathtt{p}(n) \leq 2^{2^n}$.
   By the induction hypothesis,
   $$\prod_{i=0}^{n} \mathtt{p}(i) \leq \prod_{i=0}^{n} 2^{2^i} = 2^{\sum_{i=0}^{n} 2^i} < 2^{2^{n+1}}.$$
   Then $\prod_{i=0}^{n} \mathtt{p}(i) + 1 \leq 2^{2^{n+1}}$.

   - We have two cases for the number $\prod_{i=0}^{n} \mathtt{p}(i) + 1$. It is either prime, in which case it is clear that $\mathtt{p}(n+1) \leq 2^{2^{n+1}}$, or $\prod_{i=0}^{n} \mathtt{p}(i) + 1 = \mathtt{p}(k) \cdot u$, for some number u and $k \geq n+1$. Then clearly,
   $$\mathtt{p}(n+1) \leq \mathtt{p}(k) \leq \mathtt{p}(k) \cdot u = \prod_{i=0}^{n} \mathtt{p}(i) + 1 \leq 2^{2^{n+1}}.$$

   - Now we are ready to give the primitive recursive definition of $\mathtt{p}(x)$.
   $$\left| \begin{array}{lcl} \mathtt{p}(0) & = & 2 \\ \mathtt{p}(x+1) & = & \mu z_{z \leq 2^{2^x}} [\chi_{>}(z, \mathtt{p}(x)) \cdot \mathtt{Pr}(z) = 1]. \end{array} \right.$$

7) Consider the function

$$(x)_y = \text{the least } n \text{ such that } \mathtt{p}(y)^{n+1} \nmid x.$$

For example,

$$(12)_0 = 2$$
$$(12)_1 = 1$$
$$(12)_2 = 0.$$

It is easy to see that it is primitive recursive:

$$\begin{aligned}(x)_y = &\text{ the least number } z \text{ such that } \mathtt{p}(y)^{z+1} \nmid x\\ &= \mu z_{z<x}[\ \mathtt{exp}(\mathtt{p}(y), z+1) \nmid x\ ]\\ &= \mu z_{z<x}[\ \mathtt{Div}(\mathtt{exp}(\mathtt{p}(y), z+1), x) = 0\ ].\end{aligned}$$

Problem 4. Prove that the following functions are primitive recursive:

1) $\mathtt{sq}(x) = \lfloor\sqrt{x}\rfloor$;

2) $\mathtt{lg}(x) = \lfloor\log_2(x)\rfloor$, where $\mathtt{lg}(0) = 0$;

3) $\mathtt{lcd}(x, y)$ = the least common denominator of x and y;

4) $\mathtt{gcd}(x, y)$ = the greatest common divisor of x and y;

5) $\tau(x)$ = the count of prime numbers $\leq x$;

6) $\theta(x)$ = the first prime number $\geq x$;

7) $\varphi(x)$ = the count of all numbers $\leq x$ and co-prime with x, if $x > 0$;

8) $\mathtt{len}(p, x)$ = the length of x, when x is represented in base p, $p > 1$;

9) $\mathtt{ones}(x)$ = the count of ones in the binary representation of x;

Hint.

1) Use the following representation

$$\lfloor\sqrt{x+1}\rfloor = \begin{cases}\lfloor\sqrt{x}\rfloor, & \text{if } x+1 \neq (\lfloor\sqrt{x}\rfloor + 1)^2\\ \lfloor\sqrt{x}\rfloor + 1, & \text{if } x+1 = (\lfloor\sqrt{x}\rfloor + 1)^2,\end{cases}$$

or bounded minimization

$$\begin{aligned}\lfloor\sqrt{x}\rfloor &= \text{the least } z < x \text{ such that } x < (z+1)^2\\ &= (\mu z < x)[\ \overline{\mathtt{sign}}((z+1)^2 \dot{-} x) = 0\ ].\end{aligned}$$

9) You can use the following recursive definition:

$$\left|\begin{array}{lcl}\mathtt{ones}(0) & = & 0\\ \mathtt{ones}(2x) & = & \mathtt{ones}(x)\\ \mathtt{ones}(2x+1) & = & 1 + \mathtt{ones}(2x).\end{array}\right.$$

□

1.4 Coding of finite objects

1.4.1 Coding of finite sets

We know that every natural number x can be represented in binary number system, that is, there exists numbers $0 \leq k_1 < k_2 < \cdots < k_n$ such that:

E.g. $11 = 2^3 + 2^1 + 2^0$ and its binary representation is 1011. Notice that the code of the empty set is 0, i.e. the empty sum.

$$x = 2^{k_1} + 2^{k_2} + \cdots + 2^{k_n}.$$

It follows that we can assign a code to every finite set of natural numbers. If we have the set $D = \{k_1 < k_2 < \cdots < k_n\}$, then the code of D is the number $v = 2^{k_1} + 2^{k_2} + \cdots + 2^{k_n}$. In this case, we denote the finite set as D_v.

We also have that every natural number is a code of a finite set

1) The predicate `mem` saying whether $x \in D_v$ is primitive recursive:

$$\texttt{mem}(x, v) = \begin{cases} \texttt{True}, & \text{if } x \in D_v \\ \texttt{False}, & \text{if } x \notin D_v. \end{cases}$$

Consider the finite set $D_v = \{k_0 < \cdots < k_n\}$ with code v, i.e.

$$v = 2^{k_0} + 2^{k_1} + \cdots + 2^{k_n}.$$

For a number x, we can represent v as

$$v = 2^x \cdot \texttt{qt}(2^x, v) + \texttt{rem}(2^x, v).$$

If $k_i \leq x < k_{i+1}$, we have:

$\sum_{0 \leq j < i} 2^{k_j} < 2^x$

$$v = 2^x \cdot \sum_{i \leq j \leq n} 2^{k_j - x} + \sum_{0 \leq j < i} 2^{k_j}.$$

We have two cases to consider.

- Let $x \in D_v$. It means that $x = k_i$, for some i, and hence

$$\texttt{qt}(2^x, v) = \sum_{i \leq j \leq n} 2^{k_j - k_i} = 2^0 + \sum_{i < j \leq n} 2^{k_j - k_i},$$

  which is an *odd* number, i.e.

$$\texttt{rem}(2, \texttt{qt}(2^x, v)) = 1.$$

- Let $x \notin D_v$. Then we have three sub-cases:

– if $x < k_0$, then

$$v = 2^x \sum_{0 \le i \le n} 2^{k_i - x}.$$

– if $k_i < x < k_{i+1}$, for some i, then

$$v = 2^x \sum_{i < j \le n} 2^{k_j - x} + \sum_{0 \le j \le i} 2^{k_j}.$$

– if $k_n < x$, then

$$v = 2^x \cdot 0 + \sum_{0 \le i \le n} 2^{k_i}.$$

In all three sub-cases, we have $\texttt{rem}(2, \texttt{qt}(2^x, v)) = 0$.

We conclude the $\texttt{mem}(x, v)$ is a primitive recursive function because it can be defined as

$$\texttt{mem}(x, v) \overset{\text{def}}{=} \texttt{rem}(2, \texttt{qt}(2^x, v)).$$

2) The function `card` finding the cardinality of the set D_v is primitive recursive:

$$\texttt{card}(v) = |D_v|.$$

To see why, consider the following definition:

$$\texttt{card}(v) \overset{\text{def}}{=} \sum_{x < v} \texttt{mem}(x, v).$$

3) The function `el` finding the i-th element in the set D_v is primitive recursive:

$$\texttt{el}(v, i) = \begin{cases} k_i, & \text{if } D_v = \{k_0 < \cdots < k_i < \cdots < k_n\} \\ v, & \text{otherwise} \end{cases}.$$

Consider the following definition:

$$\texttt{el}(v, i) \overset{\text{def}}{=} (\mu z < v)[\sum_{y \le z} \texttt{mem}(y, v) = i + 1].$$

4) The function `cap` finding the code of the intersection of the sets D_u and D_v is primitive recursive:

$$\texttt{cap}(u, v) = w \ \Leftrightarrow \ D_u \cap D_v = D_w.$$

Consider the following definition:

$$\texttt{cap}(u, v) \overset{\text{def}}{=} \sum_{x < u} \texttt{mem}(x, u) * \texttt{mem}(x, v) * 2^x.$$

5) The function `diff` finding the code of the difference of the sets D_u and D_v is primitive recursive:

$$\texttt{diff}(u,v) = w \iff D_u \setminus D_v = D_w.$$

Consider the following definition:

$$\texttt{diff}(u,v) = \sum_{x<u} \texttt{mem}(x,u) * \overline{\texttt{sign}}(\texttt{mem}(x,v)) * 2^x.$$

6) The function `cup` finding the code of the union of the sets D_u and D_v is primitive recursive:

$$\texttt{cup}(u,v) = w \iff D_u \cup D_v = D_w.$$

Consider the following definition:

$$\texttt{cup}(u,v) \overset{\text{def}}{=} \texttt{diff}(u,v) + v.$$

7) The predicate `subs` saying whether D_u is a subset of D_v is primitive recursive: ✍ Homework!

$$\texttt{subs}(u,v) = \begin{cases} 1, & \text{if } D_u \subseteq D_v \\ 0, & \text{otherwise.} \end{cases}$$

8) The function `power` finding the code of the powerset of the set D_u is primitive recursive: ✍ Homework!

$$\texttt{power}(u) = v,$$

where $D_v = \{x \mid D_x \subseteq D_u\}$.

1.4.2 Coding of finite sequences

We say that the functions π, λ, ρ form a **coding triple** if they satisfy the properties:

$$\pi(\lambda(z), \rho(z)) = z, \quad \lambda(\pi(x, y)) = x, \quad \rho(\pi(x, y)) = y.$$

It is easy to see that if $\langle \pi, \lambda, \rho \rangle$ is a coding triple, then $\pi : \mathbb{N}^2 \to \mathbb{N}$ is bijective.

Proposition 1.1. There exists a primitive recursive coding triple $\langle \pi, \lambda, \rho \rangle$ with the property $\lambda(z) \leq z$ and $\rho(x) \leq z$.

Important property for bounded minimization

Proof. There are many such triples. Here we use the Cantor coding:

$$\pi(x, y) \overset{\text{def}}{=} \sum_{i=1}^{x+y} i + y = \frac{(x + y + 1)(x + y)}{2} + y.$$

We need the primitive recursive function

$$\omega(z) \overset{\text{def}}{=} (\mu s \leq z)[\ \sum_{i=1}^{s+1} i < z\ ].$$

It is easy to see that we have the property:

$$\omega(\pi(x, y)) = x + y.$$

Then we define the decoding functions in the following way:

$$\lambda(z) \overset{\text{def}}{=} \omega(z) - \rho(z)$$
$$\rho(z) \overset{\text{def}}{=} z - \sum_{i \leq w(z)} i.$$

$\square$

Given a coding triple $\langle \pi, \lambda, \rho \rangle$, we show that we can build, for every $k \geq 1$, a coding $(k + 1)$-tuple of functions $\langle \pi_k, J_1^k, \ldots, J_k^k \rangle$. We do this inductively by starting from $k = 1$ and define a coding $(k + 1)$-tuple using the functions from the coding k-tuple.

Change the notation J_k^n.

Notice that the definition of π_{k+1} is not how it is done is Lisp-like programming languages with the atomic operations of `cons`, `car` and `cdr`.

- $\pi_1(x) \overset{\text{def}}{=} x$ and $J_1^1(x) \overset{\text{def}}{=} x$.

- $\pi_{k+1}(x_1, \ldots, x_{k+1}) \overset{\text{def}}{=} \pi(\pi_k(x_1, \ldots, x_k), x_{k+1})$.
  $J_i^{k+1}(z) \overset{\text{def}}{=} J_i^k(\lambda(z))$, for $i = 1, \ldots, k$, and $J_{k+1}^{k+1}(z) \overset{\text{def}}{=} \rho(z)$.

Proposition 1.2. Let us have fixed a primitive recursive coding triple $\langle \pi, \lambda, \rho \rangle$. Then for every $k \geq 1$, the functions π_k and J_i^k are primitive recursive.

Let us denote $\mathbb{N}^+ = \bigcup_{k>0} \mathbb{N}^k$. Now we define $\tau : \mathbb{N}^+ \to \mathbb{N}$ in the following way:

$$\tau(x_0, \ldots, x_n) \overset{\text{def}}{=} \pi(n, \pi_{n+1}(x_0, \ldots, x_n)).$$

Problem 5. Prove the following:

1) τ is bijective;

2) the function `len` finding the length of the tuple with code z primitive recursive: $\texttt{len}(z) = k$ iff z is the code of a k-tuple.

$\texttt{len}(z) = \lambda(z) + 1$

3) the function `mem` finding the i-th member of the tuple with code z is primitive recursive:

$$\texttt{mem}(z, i) = \begin{cases} a_i, & \text{if } z \text{ is the code of } \langle a_0, \ldots, a_i, \ldots, a_k \rangle \\ z, & \text{otherwise.} \end{cases}$$

Hint. Let us consider the following function:

Recall that:

$$f^0(x) = x$$
$$f^{k+1}(x) = f(f^k(x))$$

$$G(k, i, z) = \begin{cases} \lambda^k(z), & \text{if } i = 0, \\ \rho(\lambda^{k-i}(z)), & \text{if } 1 \leq i \leq k, \\ z, & \text{otherwise} \end{cases}$$

It is easy to check that since λ and ρ are primitive recursive, G is primitive recursive and have the property that $G(k, i, z) = J_i^k(z)$. Thus,

$$\texttt{mem}(z, i) \overset{\text{def}}{=} G(\lambda(z), i, \rho(z)).$$

□

Problem 6. Let us consider the function pairing function:

✍ Homework!

$$\pi(x, y) \overset{\text{def}}{=} 2^x(2y + 1) - 1.$$

- Show that π is primitive recursive and bijective.
- Show that there exist primitive recursive function λ and ρ such that $\langle \pi, \lambda, \rho \rangle$ is a coding triple with the property $\lambda(z) \leq z$ and $\rho(z) \leq z$.

Problem 7. Let us consider the function $\tau : \mathbb{N}^+ \to \mathbb{N}$ defined as: ✍ Homework!

$$\tau(a_0, \dots, a_k) \stackrel{\text{def}}{=} 2^{a_0+0} + 2^{a_0+a_1+1} + \dots + 2^{a_0+a_1+\dots+a_k+k} - 1.$$

Show that the function τ is bijective and prove that the following functions are primitive recursive:

1) $\texttt{len}(v) =$ the length of $\bar{a}$, where $v = \tau(\bar{a})$.

2) $\texttt{mem}(v, i) = \begin{cases} a_i, & \text{if } v = \tau(a_1, \dots, a_i, \dots, a_k) \\ v, & \text{if } \texttt{len}(v) < i. \end{cases}$

3) $\texttt{pref}(u, v) = \begin{cases} \texttt{True}, & u = \tau(a_0, \dots, a_{k-1}) \ \& \ v = \tau(a_0, \dots, a_{k-1}, \dots, a_{m-1}) \\ \texttt{False}, & \text{otherwise} \end{cases}$

In the next problem we use the coding functions defined in the previous two problems.

Problem 8. The set T of natural numbers is called a *tree* if ✍ Homework!

$$(\forall u, v \in \mathbb{N})[(v \in T \ \& \ \texttt{pref}(u, v) = \texttt{True}) \implies v \in T].$$

Show that the function

$$\texttt{tree}(v) = \begin{cases} \texttt{True}, & \text{if } D_v \text{ is a tree} \\ \texttt{False}, & \text{if } D_v \text{ is not a tree} \end{cases}$$

is primitive recursive.

1.5 Additional schemes

1.5.1 Simultaneous recursion

We say that f and g are obtained by **simultaneous recursion** from f_0, g_0, F and G, if

[7] p. 61

$$\left|\begin{array}{lcl} f(x,0) & = & f_0(x) \\ g(x,0) & = & g_0(x) \\ f(x,n+1) & = & F(x,n,f(x,n),g(x,n)) \\ g(x,n+1) & = & G(x,n,f(x,n),g(x,n)). \end{array}\right.$$

We will show that the class of primitive recursive functions is closed under the scheme for simultaneous recursion.

It can be generalised for any finite number of functions

Theorem 1.3. If f_0, g_0, F and G are primitive recursive, then so are f and g.

Proof. Let us fix the coding triple $\langle \pi, \lambda, \rho \rangle$. We will show that the following function is primitive recursive:

$$h(x,n) = \pi(f(x,n), g(x,n)).$$

It will follow that f and g are primitive recursive, because

$$\begin{aligned} f(x,n) &= \lambda(h(x,n)) \\ g(x,n) &= \rho(h(x,n)). \end{aligned}$$

We have the following property:

$$\begin{aligned} h(x,n+1) &= \pi(F(x,n,f(x,n),g(x,n)), G(x,n,f(x,n),g(x,n))) \\ &= \pi(F(x,n,\lambda(h(x,n)),\rho(h(x,n))), G(x,n,\lambda(h(x,n)),\rho(h(x,n)))). \end{aligned}$$

Since we have the primitive recursive function

$$H(x,n,y) = \pi(F(x,n,\lambda(y),\rho(y)), G(x,n,\lambda(y),\rho(y))),$$

we see that h is primitive recursive, because its definition follows the scheme:

$$\left|\begin{array}{lcl} h(x,0) & \overset{\text{def}}{=} & \pi(f_0(x), g_0(x)) \\ h(x,n+1) & \overset{\text{def}}{=} & H(x,n,h(x,n)). \end{array}\right.$$

$\square$

1.5.2 Course-of-values recursion

Let us consider two examples of computable functions. The first gives us the famous Fibonacci sequence.

[5, p. 89]. It starts with $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$

$$\mathtt{fib}(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ \mathtt{fib}(n-2) + \mathtt{fib}(n-1), & n \geq 2 \end{cases}$$

The second one defines the function x^y.

$$\mathtt{pow}(x, y) = \begin{cases} 1 & y = 0 \\ x * \mathtt{pow}(x, y-1), & y > 0 \text{ \& } y \text{ is odd} \\ \mathtt{pow}(x, y/2)^2, & y > 0 \text{ \& } y \text{ is even.} \end{cases}$$

Their definitions follow a pattern similar to the primitive recursive scheme, but to compute the next value of the function, they need to know more than just the last computed value.

Problem 9. Prove that the function `fib` is primitive recursive.

Proof. Let us fix the coding triple $\langle \pi, \lambda, \rho \rangle$. We have the primitive recursive function f:

$$\left|\begin{array}{lcl} f(0) & = & \pi(0, 1) \\ f(n+1) & = & \pi(\rho(f(n)), \lambda(f(n)) + \rho(f(n))). \end{array}\right.$$

Then $\mathtt{fib}(n) = \lambda(f(n))$ □

Problem 10. Prove that the function $pow(x, y)$ is primitive recursive.

Lemma 1.1. Let f be primitive recursive. Then the function

$$H_f(\overline{x}, y) = \tau(f(\overline{x}, 0), f(\overline{x}, 1), \ldots, f(\overline{x}, y))).$$

is primitive recursive. We call H_f the **history** of f.

Proof. Following the definition of τ in .... , we have the property:

$$H_f(\overline{x}, y+1) = \pi(y+1, \pi(\rho(H_f(\overline{x}, y)), f(\overline{x}, y+1))).$$

Let us denote

$$F(\overline{x}, y, z) = \pi(y+1, \pi(\rho(z), f(\overline{x}, y+1)),$$

which is clearly primitive recursive. Then we see that H_f is produced from π and F by the primitive recursive scheme:

$$\left| \begin{array}{lcl} H_f(\overline{x}, 0) & = & \pi(0, f(\overline{x}, 0)) \\ H_f(\overline{x}, y+1) & = & F(\overline{x}, y, H_f(\overline{x}, y)). \end{array} \right.$$

□

Recall how we proved that the function `fib` is primitive recursive. Using our fixed primitive recursive coding triple $\langle \pi, \lambda, \rho \rangle$, we will elaborate on that idea and prove a general result.

We say that the function f is obtained by **course-by-values recursion**, if it follows the scheme:

$$\left| \begin{array}{lcl} F(\overline{x}, 0) & = & f(\overline{x}) \\ F(\overline{x}, y+1) & = & g(\overline{x}, y, H_F(\overline{x}, y)) \end{array} \right.$$

Theorem 1.4. Let f and g are primitive recursive and F is obtained from f and g by course-by-values recursion. Then F is primitive recursive as well.

In other words, the class of primitive recursive functions is closed under course-by-values recursion.

Proof. It is enough to prove that H_F is primitive recursive. We have:

$$\begin{aligned} H_F(\overline{x}, y+1) &= \tau(F(\overline{x}, 0), \dots, F(\overline{x}, y+1)) \\ &= \pi(y+1, \pi(\rho(H_F(\overline{x}, y)), F(\overline{x}, y+1))) \\ &= \pi(y+1, \pi(\rho(H_F(\overline{x}, y)), g(\overline{x}, y, H_F(\overline{x}, y))) \end{aligned}$$

Thus, H_F is primitive recursive, because it follows the primitive recursive scheme:

$$\left| \begin{array}{lcl} H_F(\overline{x}, 0) & = & \pi(0, f(\overline{x}, 0)) \\ H_F(\overline{x}, y+1) & = & \pi(y+1, \pi(\rho(H_F(\overline{x}, y)), g(\overline{x}, y, H_F(\overline{x}, y))). \end{array} \right.$$

Now, it follows that F is primitive recursive, because:

$$F(\overline{x}, y) = \begin{cases} f(\overline{x}), & \text{if } y = 0 \\ g(\overline{x}, y \dot{-} 1, H_F(\overline{x}, y \dot{-} 1)), & \text{if } y > 0. \end{cases}$$

□

We give a second proof of the fact that `fib` is primitive recursive.

Problem 11. The function `fib` is primitive recursive, where

$$\mathtt{fib}(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ \mathtt{fib}(n-2) + \mathtt{fib}(n-1), & n \geq 2. \end{cases}$$

Proof. Consider the primitive recursive function

We use `mem` from Problem 5.

$$\left| \begin{array}{lcl} g(0,z) & \overset{\text{def}}{=} & 1 \\ g(n+1,z) & \overset{\text{def}}{=} & \mathtt{mem}(z, n \dot{-} 1) + \mathtt{mem}(z,n). \end{array} \right.$$

Then we have:

$$\left| \begin{array}{lcl} \mathtt{fib}(0) & \overset{\text{def}}{=} & 0 \\ \mathtt{fib}(n+1) & \overset{\text{def}}{=} & g(n, H_{\mathtt{fib}}(n)). \end{array} \right.$$

By *Lemma* 1.1, `fib` is primitive recursive. □

Similarly, we can apply *Lemma* 1.1 to prove that function `pow` is primitive recursive. Now we will state a generalisation of this result.

✍ Do it!

Problem 12. Let f, r and s are primitive recursive. Prove that

$$h(\overline{x}, y) = \begin{cases} f(\overline{x}, y, h(\overline{x}, r(y))), & \text{if } r(y) < y \\ s(\overline{x}, y), & \text{otherwise} \end{cases}$$

is primitive recursive.

1.5.3 Tail recursion

Compare with 4).

Problem 13. Let $f : \mathbb{N}^2 \to \mathbb{N}$ be a primitive recursive function. Show that the function

$$\left| \begin{array}{lcl} f^\star(0,r) & = & r \\ f^\star(n+1,r) & = & f^\star(n, f(n,r)) \end{array} \right.$$

is primitive recursive.

Hint. Let us start with an example:

$$f^\star(4,r) = f(0, f(1, f(2, f(3,r)))).$$

Consider the function $\hat{f}$ with the following *primitive recursive* definition:

✍ Explain why this definition follows the primitive recursive sheme

$$\left| \begin{array}{lcl} \hat{f}(0,k,r) & \overset{\text{def}}{=} & r \\ \hat{f}(\ell+1,k,r) & \overset{\text{def}}{=} & f(k \dot{-} (\ell+1), \hat{f}(\ell,k,r)). \end{array} \right.$$

Finally, the function $f^\star$ is primitive recursive because

$$f^\star(\ell, r) = \hat{f}(\ell, \ell, r).$$

□

Show the following:

$\hat{f}(2,2,r) = f(0, f(1,r));$
$\hat{f}(2,3,r) = f(1, f(2,r));$
$\hat{f}(3,2,r) = f(0, f(0, f(1,r))).$

Problem 14. Let the function f be defined in the following way:

$$\left| \begin{array}{lcl} f(0,x) & \overset{\text{def}}{=} & g(x) \\ f(n+1,x) & \overset{\text{def}}{=} & f(n, p(n,x)). \end{array} \right.$$

Show that if g and p are primitive recursive, then f is primitive recursive.

Proof. To get an idea about how to proceed with the proof, let us start by calculating $f(n,x)$ for the first few values of n:

$$\begin{aligned} f(0,x) &= g(x) \\ f(1,x) &= g(p(0,x)) \\ f(2,x) &= g(p(0,p(1,x))) \\ f(3,x) &= g(p(0,p(1,p(2,x)))) \\ &\vdots \end{aligned}$$

It follows that

$$f(n,x) = g(p^\star(n,x))$$

and hence it is primitive recursive. □

Example 1. We know that we can define the factorial function $x!$ in the following way:

$$\left| \begin{array}{lcl} \texttt{fact}(0,r) & = & r \\ \texttt{fact}(n+1,r) & = & \texttt{fact}(n, (n+1) * r). \end{array} \right.$$

Clearly, $x! = \texttt{fact}(x,1)$. It follows from *Problem* 14 that the function `fact` is primitive recursive.

1.6 A function, which is not primitive recursive

The main idea here is to build a recursive function which grows faster than any primitive recursive function. Let us start a sequence of primitive recursive functions in the following way:

This section is based on [7, p. 105].

$$
\begin{aligned}
\psi_0(n,a) &= a+n\\
\psi_1(n,a) &= a\cdot n\\
\psi_2(n,a) &= a^n\\
\psi_3(n,a) &= a^{a^{\cdot^{\cdot^{\cdot^{a}}}}}\\
&\vdots
\end{aligned}
$$

How do we continue this sequence? We will try to find a pattern in the primitive recursive definition of the functions above. Let us recall the primitive recursive definition of ψ_1:

$$
\left|\begin{array}{lcl}
\psi_1(0,a) & = & 0\\
\psi_1(n+1,a) & = & \psi_0(\psi_1(n,a),a),
\end{array}\right.
$$

and that of ψ_2:

$$
\left|\begin{array}{lcl}
\psi_2(0,a) & = & 1\\
\psi_2(n+1,a) & = & \psi_1(\psi_2(n,a),a)
\end{array}\right.
$$

We want to define $\psi_3(n,a)$ so that we follow the pattern that $\psi_3(n+1,a) = \psi_2(\psi_1(n,a),a)$. It turns out that $\psi_3(n,a)$ will be the n-th iteration of the raising to a power of a. For instance,

$$\psi_3(3,a) = a^{a^{a^{a}}}.$$

We can define ψ_3 by using the primitive recursive scheme:

$$
\left|\begin{array}{lcll}
\psi_3(0,a) & = & \psi_2(1,a) & //\ =a\\
\psi_3(n+1,a) & = & \psi_2(\psi_3(n,a),a) & //\ =a^{\psi_3(n,a)}
\end{array}\right.
$$

In general, for $m \geq 2$, we want $\psi_{m+1}(n,a)$ to be the n-th iteration of ψ_m, i.e.

$$
\left|\begin{array}{lcl}
\psi_{m+1}(0,a) & = & \psi_m(1,a)\\
\psi_{m+1}(n+1,a) & = & \psi_m(\psi_{m+1}(n,a),a).
\end{array}\right.
$$

In this way, we build an infinite sequence $P = \{\psi_n \mid n \in \mathbb{N}\}$ of primitive recursive functions in which each function in the sequence grows much faster than the previous function. We can define a ternary function Ψ which enumerates the sequence P.

This is roughly what Wilhelm Ackermann did in 1928. The simplified version was given by Rozsa Péter and Robinson.

$$\begin{aligned}
&\Psi(0,n,a) = n+a \\
&\Psi(1,0,a) = 0 \\
&\Psi(2,0,a) = 1 \\
&\Psi(m+1,0,a) = \Psi(m,1,a), && \text{if } m \geq 2 \\
&\Psi(m+1,n+1,a) = \Psi(m,\Psi(m+1,n,a),a), && \text{if } m \geq 1.
\end{aligned}$$

For our purposes, we do not need to work with this complicated function Ψ. What is important for us is the primitive recursive scheme, because it shows how we obtain new functions in the sequence from the old ones. Omitting the third argument, we obtain:

$$\left|\begin{array}{lcl} \psi(m+1,0) &=& \psi(m,1) \\ \psi(m+1,n+1) &=& \psi(m,\psi(m+1,n)). \end{array}\right.$$

It remains to define the function ψ when the first argument is 0. In the following proof, we will need the property that $\psi(m,n) > n$, so we define $\psi(0,n) = n+1$.

Theorem 1.5. Let us consider the function ψ given by double recursion:

$$\left|\begin{array}{lcl} \psi(0,n) &=& n+1 \\ \psi(m+1,0) &=& \psi(m,1) \\ \psi(m+1,n+1) &=& \psi(m,\psi(m+1,n)) \end{array}\right.$$

The function ψ is **not** primitive recursive.

We divide the proof into several steps.

Proposition 1.3. The function ψ is total.

Hint. Easy induction on the lexicographical order. □

Proposition 1.4. $(\forall m,n \in \mathbb{N})[\psi(m,n) \geq n+1]$.

Proof. For $m = 0$ it follows from the definition. As induction hypothesis, suppose $(\forall n)[\psi(m,n) \geq n+1]$. For $m+1$, we do an induction on n. For $n = 0$,

$$\psi(m+1,0) = \psi(m,1) \geq 1 = 0+1.$$

For $n > 0$,

$$\begin{aligned}
\psi(m+1,n) &= \psi(m,\psi(m+1,n-1)) && \text{// by def.} \\
&\geq \psi(m+1,n-1)+1 && \text{// by I.H. for } m \\
&\geq n+1. && \text{// by I.H. for } n
\end{aligned}$$

□

Proposition 1.5. The function ψ increases monotonically on the second argument, i.e.

$$(\forall m, n \in \mathbb{N})[\psi(m, n) < \psi(m, n + 1)].$$

Proof. Induction on m. For $m = 0$, it follows from the definition of ψ. For the induction step,

$$\begin{aligned} \psi(m+1, n+1) &= \psi(m, \psi(m+1, n)) && \text{// by def.} \\ &\geq \psi(m+1, n) + 1 && \text{// by } \textit{Proposition } 1.4 \\ &> \psi(m+1, n). \end{aligned}$$

□

Proposition 1.6. $(\forall m, n \in \mathbb{N})[\psi(m + 1, n) \geq \psi(m, n + 1)]$.

Proof. Induction on n. For $n = 0$, we have $\psi(m + 1, 0) = \psi(m, 1)$. For the induction step,

$$\begin{aligned} \psi(m+1, n+1) &= \psi(m, \psi(m+1, n)) && \text{// by def.} \\ &\geq \psi(m, \psi(m, n+1)) && \text{// by I.H. and } \textit{Proposition } 1.5 \\ &\geq \psi(m, n+2). && \text{// by } \textit{Proposition } 1.4 \text{ and } \textit{Proposition } 1.5 \end{aligned}$$

□

Proposition 1.7. The function ψ increases monotonically on the first argument, i.e.

$$(\forall m \in \mathbb{N})(\forall n \in \mathbb{N})[\psi(m, n) < \psi(m + 1, n)].$$

Proof. For any m and n, we have the following:

$$\begin{aligned} \psi(m+1, n) &\geq \psi(m, n+1) && \text{// by } \textit{Proposition } 1.6 \\ &> \psi(m, n). && \text{// by } \textit{Proposition } 1.5 \end{aligned}$$

□

Proposition 1.8. For every primitive recursive function g, there exists k such that

$$(\forall x_1, \ldots, x_n \in \mathbb{N})[g(x_1, \ldots, x_n) < \psi(k, \max\{x_1, \ldots, x_n\})].$$

Proof. We proceed by induction on the definition of primitive recursive functions.

- It is easy when g is some of $\mathtt{O}(x)$, $\mathtt{S}(x)$, $\mathtt{I}^n_i(\bar{x})$.

- Let $g(\bar{x}) = h(f_1(\bar{x}), \ldots, f_m(\bar{x}))$. By I.H. we have k_i is such that $f_i(\bar{x}) < \psi(k_i, x)$ and ℓ is such that $h(\bar{x}, y, z) < \psi(\ell, \max\{\bar{x}, y, z\})$. Let

$$k = \max\{k_1, \ldots, k_m, \ell\}.$$

By monotonicity, that is *Proposition* 1.7, $f_i(\bar{x}) < \psi(k, x)$. Clearly,

$$\max\{f_1(\bar{x}), \ldots, f_m(\bar{x})\} < \psi(k, x).$$

Then we have the following:

$$\begin{aligned} g(\bar{x}) &= h(f_1(\bar{x}), \ldots, f_m(\bar{x})) && \text{// by def. of } g \\ &< \psi(k, \max\{f_1(\bar{x}), \ldots, f_m(\bar{x})\}) && \text{// by the choice of } k \\ &< \psi(k, \psi(k, x)) && \text{// by } \textit{Proposition } 1.5 \\ &< \psi(k, \psi(k+1, x)) && \text{// by } \textit{Proposition } 1.5 \text{ and } \textit{Proposition } 1.7 \\ &= \psi(k+1, x+1) && \text{// by def. of } \psi \\ &\leq \psi(k+2, x) && \text{// by } \textit{Proposition } 1.6. \end{aligned}$$

- Let g be defined by primitive recursion:

$$\left| \begin{array}{lcl} g(\bar{x}, 0) & = & f(\bar{x}), \\ g(\bar{x}, n+1) & = & h(\bar{x}, n, g(\bar{x}, n)). \end{array} \right.$$

By I.H., we have a number ℓ such that $f(\bar{x}) < \psi(\ell, x)$ and and r such that $h(\bar{x}, i, z) < \psi(r, \max\{\bar{x}, i, z\})$. Let $k = \max\{\ell, r\}$. We shall find a number m such that

Clearly m will be $\geq k$

$$(\forall \bar{x})(\forall i)[g(\bar{x}, i) < \psi(m, \max\{\bar{x}, i\})].$$

Our first step is to prove by induction on i that

$$(\forall i)(\forall \bar{x})[g(\bar{x}, i) < \psi(k+1, \max\{\bar{x}\} + i)]. \tag{1.1}$$

For $i = 0$, it is clear, since $\max\{\bar{x}\} + 0 = x = \max\{\bar{x}, 0\}$. For the induction step,

$$\begin{aligned} g(\bar{x}, i+1) &= h(\bar{x}, i, g(\bar{x}, i)) && \text{// by def. of } g \\ &< \psi(k, \max\{\bar{x}, i, g(\bar{x}, i)\}) && \text{// by the choice of } k \\ &< \psi(k, \max\{\bar{x}, i, \psi(k+1, \max\{\bar{x}\} + i)\}) && \text{// by I.H.} \\ &= \psi(k, \psi(k+1, \max\{\bar{x}\} + i)) \\ &= \psi(k+1, \max\{\bar{x}\} + i + 1) && \text{// by def. of } \psi. \end{aligned}$$

Now we are ready to finish the proof. We claim that

$$(\forall i)(\forall \bar{x})[g(\bar{x}, i) < \psi(k+3, \max\{\bar{x}, i\})].$$

By an easy induction:

$$\psi(1,n) = n+2$$
$$\psi(2,n) = 2n+3$$

This is easy:

$$\begin{aligned}
g(\overline{x}, i) &< \psi(k+1, \max\{\bar{x}\} + i) && \text{// by 1.1}\\
&< \psi(k+1, 2\max\{\bar{x}, i\} + 3) && \text{// by monotonicity}\\
&= \psi(k+1, \psi(2, \max\{\bar{x}, i\})) && \text{// by def.}\\
&< \psi(k+1, \psi(k+2, \max\{\bar{x}, i\})) && \text{// by monotonicity}\\
&= \psi(k+2, \max\{\bar{x}, i\} + 1) && \text{// by def.}\\
&\le \psi(k+3, \max\{\bar{x}, i\}) && \text{// by \textit{Proposition} 1.6.}
\end{aligned}$$

□

Problem 15. Let us consider the function φ given by double recursion:

$$\left|\begin{array}{lcl}
\varphi(0,n) &=& n\\
\varphi(m+1,0) &=& \varphi(m,1)\\
\varphi(m+1,n+1) &=& \varphi(m, \varphi(m+1,n))
\end{array}\right.$$

Show that the function φ has the following form:

$$\varphi(x,y) = \begin{cases} y, & \text{if } x = 0\\ 1, & \text{otherwise.}\end{cases}$$

Iteration

Definition 1.2. Given a function f, the function $f^\star$ is defined by iteration from f if $f^\star(x,n) = f^{(n)}(x)$.

Recall that

$$\left|\begin{array}{lcl}
f^{(0)}(x) &\stackrel{\text{def}}{=}& x\\
f^{(n+1)}(x) &\stackrel{\text{def}}{=}& f(f^{(n)}(x)).
\end{array}\right.$$

Proposition 1.9 (Robinson, Bernays)**.** The class of primitive recursive functions is the smallest class of functions

- containing the initial functions, together with coding and decoding functions for pairs;

- closed under composition;

- closed under iteration.

Hint. See [5, p. 72] or [6, p. 295]. □

Theorem 1.6. Let us consider the following sequence of functions

$$\left|\begin{array}{lcl}\psi_0(x) & = & x+1 \\ \psi_{n+1}(x) & = & \psi_n^{(x)}(x).\end{array}\right.$$

Then:

1) ψ_n is primitive recursive, for each n;

2) for every primitive recursive function f, there is an index n such that $f(\overline{x}) \leq \psi_n(\sum \overline{x})$ for almost every $\overline{x}$.

3) the diagonal function $d(x) = \psi_x(x)$ dominates every primitive recursive function, and thus is not primitive recursive.

Hint. See [6, p. 298]. □

Chapter 2

Unlimited Register Machines

2.1 Description of the machine

The main sources here are [2] and [1]. The unlimited register machines (URM) were introduced by Shepherdson and Sturgis [9].

- We have an infinite array of registers, which we denote by

$$\mathtt{r}[1], \mathtt{r}[2], \ldots, \mathtt{r}[n], \ldots$$

- We also have four types of **instructions** for URM:
  - $\mathtt{Zero}(n)$, $n \in \mathbb{N}$, $n \geq 1$, with meaning $\mathtt{r}[n] := 0$;
  - $\mathtt{Succ}(n)$, $n \in \mathbb{N}$, $n \geq 1$, with meaning $\mathtt{r}[n] := \mathtt{r}[n] + 1$;
  - $\mathtt{Set}(m, n)$, $n, m \in \mathbb{N}$, $n, m \geq 1$, with meaning $\mathtt{r}[n] := \mathtt{r}[m]$;
  - $\mathtt{Jump}(m, n, q)$, $n, m, q \in \mathbb{N}$, $n, m \geq 1$, with meaning that if $\mathtt{r}[n] = \mathtt{r}[m]$, then we go to the q-th command, otherwise we go to the next command;

- **A program** for the language of URM is a finite sequence $\mathtt{P}$ of instructions. Usually we shall denote a program as

$$\mathtt{P} = \langle \mathtt{I}^{\mathtt{P}}_0, \ldots, \mathtt{I}^{\mathtt{P}}_{n-1} \rangle.$$

  Let $\mathtt{len}(\mathtt{P})$ be the number of instructions in the program $\mathtt{P}$.

- It is possible for a program $\mathtt{P}$ to contain an instruction of the form $\mathtt{Jump}(k, \ell, q)$, where $q \geq \mathtt{len}(P)$. We say that a program $\mathtt{P}$ is in **standard form** if every instruction $\mathtt{I}^{\mathtt{P}}_j$ of the form $\mathtt{Jump}(k, \ell, q)$ is such that $q \leq \mathtt{len}(\mathtt{P})$.

Not every program is in standard form, but every program can be converted to one in standard form preserving the computational semantics.

- Notice that it is not allowed to have programming constructions of the form $\mathtt{r}[\mathtt{r}[m]]$. It means that by just looking at the program $\mathtt{P}$, we know the indices of all registers used when running the program $\mathtt{P}$. The **rank** of $\mathtt{P}$

is the maximum of all register indices which are used in any computation of P. We shall denote the rank of P by rank(P). Notice that the rank is always finite.

- A **state** is every infinite sequence of natural numbers

$$\sigma = (\ell, a_1, a_2, \dots)$$

such that $(\exists i_0)(\forall i > i_0)[\sigma\texttt{[}i\texttt{]} = 0]$. Since we know that all but finitely many elements of σ are non-zero, we will usually denote the state σ by $(\ell, \overline{a})$, where $\overline{a} = a_1, a_2, \dots, a_{i_0}$.

The interpretation of a state σ will be that $\sigma\texttt{[0]} = \ell$ shows the index of the instruction to be executed next, and $\sigma\texttt{[}i\texttt{]} = \texttt{r[}i\texttt{]}$, for $i \geq 1$, represents the contents of the registers.

- Let us denote the set of all states by State.

- We define **the semantics** of every instruction I as the function

$$[\![\texttt{I}]\!] : \texttt{State} \to \texttt{State},$$

which returns the result of applying the instruction I on the state σ. Suppose that $\sigma = (\ell, a_1, a_2, \dots)$. Then:

$$[\![\texttt{I}]\!](\sigma) = \begin{cases} (\ell+1, a_1, \dots, a_{n-1}, \mathbf{0}, a_{n+1}, \dots), & \text{if } \texttt{I} = \texttt{Zero}(n) \\ (\ell+1, a_1, \dots, \boldsymbol{a_n} + \mathbf{1}, a_{n+1}, \dots), & \text{if } \texttt{I} = \texttt{Succ}(n) \\ (\ell+1, a_1, \dots, a_{n-1}, \boldsymbol{a_m}, a_{n+1}, \dots), & \text{if } \texttt{I} = \texttt{Set}(m,n) \\ (\ell+1, a_1, a_2, \dots), & \text{if } \texttt{I} = \texttt{Jump}(m,n,q) \;\&\; a_m \neq a_n \\ (\boldsymbol{q}, a_1, a_2, \dots), & \text{if } \texttt{I} = \texttt{Jump}(m,n,q) \;\&\; a_m = a_n \end{cases}$$

- The final step is to define what we mean by the semantics of the whole program P. We will explain what is **the result of executing** the program P for the fixed number of s steps over some state σ. We will denote this function by

$$[\![\texttt{P}]\!]_s : \texttt{State} \to \texttt{State}.$$

For zero steps, we haven't done anything yet, so we set

$$[\![\texttt{P}]\!]_0(\sigma) \overset{\text{def}}{=} \sigma.$$

Suppose we have defined $[\![\mathtt{P}]\!]_s(\sigma) = \sigma'$ and $\ell = \sigma'\mathtt{[0]}$. We will define $[\![\mathtt{P}]\!]_{s+1}(\sigma)$.

If ℓ points to a valid instruction, then we execute it. Otherwise, we do nothing.

$$[\![\mathtt{P}]\!]_{s+1}(\sigma) \overset{\text{def}}{=} \begin{cases} [\![\mathtt{I}^{\mathtt{P}}_{\ell}]\!](\sigma'), & \text{if } \ell < \mathtt{len}(\mathtt{P}) \\ \sigma', & \text{if } \ell \geq \mathtt{len}(\mathtt{P}). \end{cases}$$

For a fixed n, consider n-tuples $\overline{a}$ and $\overline{b}$. A state of the form $\sigma = (i, \overline{b})$ is called **final** for $\mathtt{P}$ on input data $\overline{a}$, if for the initial state $\sigma_0 = (0, \overline{a})$, there exists a step s for which $[\![\mathtt{P}]\!]_s(\sigma_0) = \sigma$ and $\sigma = (\ell, b_1, b_2 \dots)$, where $\ell \geq \mathtt{len}(\mathtt{P})$. In this case we write

By definition, for every step $t \geq s$, we have $[\![\mathtt{P}]\!]_t(\sigma_0) = \sigma$.

$$[\![\mathtt{P}]\!]^{(n)}(\overline{a}) \downarrow= b_1.$$

Let $f : \mathbb{N}^n \to \mathbb{N}$ be a (partial) function and $\mathtt{P}$ a program. We say that f is **URM-computable** by $\mathtt{P}$ if for every n-tuple $\overline{x}$, we have

$$[\![\mathtt{P}]\!]^{(n)}(\overline{x}) \simeq f(\bar{x}).$$

This means that $[\![\mathtt{P}]\!]^{(n)}(\bar{x}) \downarrow= y$ if and only if $f(\bar{x}) \downarrow= y$.

Lemma 2.1. For every program $\mathtt{P}$, there exists a program $\mathtt{S}$ in standard form, which is equivalent to $\mathtt{P}$, i.e.

$$(\forall \overline{a}, b)[\ [\![\mathtt{P}]\!]^{(n)}(\overline{a}) \simeq b \ \Leftrightarrow \ [\![\mathtt{S}]\!]^{(n)}(\overline{a}) \simeq b\ ].$$

Proof. Let $\mathtt{P} = \langle \mathtt{I}^{\mathtt{P}}_0, \dots, \mathtt{I}^{\mathtt{P}}_{n-1} \rangle$. We define the program $\mathtt{S} = \langle \mathtt{I}^{\mathtt{S}}_0, \dots, \mathtt{I}^{\mathtt{S}}_{n-1} \rangle$ such that for every $\ell < n$, we have the following:

$$\mathtt{I}^{\mathtt{S}}_{\ell} = \begin{cases} \mathtt{I}^{\mathtt{P}}_{\ell}, & \text{if } \mathtt{I}^{\mathtt{P}}_{\ell} \text{ is arithmetical} \\ \mathtt{I}^{\mathtt{P}}_{\ell}, & \text{if } \mathtt{I}^{\mathtt{P}}_{\ell} = \mathtt{Jump}(i, j, q) \ \& \ q \leq \mathtt{len}(\mathtt{P}) \\ \mathtt{Jump}(i, j, \mathtt{len}(\mathtt{P})), & \text{if } \mathtt{I}^{\mathtt{P}}_{\ell} = \mathtt{Jump}(i, j, q) \ \& \ q > \mathtt{len}(\mathtt{P}) \end{cases}$$

It is clear that $\mathtt{S}$ is in standard form. □

2.2 Concatenation of programs

We define the function

$$\mathtt{shift}(\mathtt{I}, s) = \begin{cases} \mathtt{I}, & \text{if } \mathtt{I} \text{ is an arithmetical insturction} \\ \mathtt{Jump}(i, j, q+s), & \text{if } \mathtt{I} = \mathtt{Jump}(i, j, q). \end{cases}$$

Let $\mathtt{P} = \langle \mathtt{I}_0^{\mathtt{P}}, \dots, \mathtt{I}_{m-1}^{\mathtt{P}} \rangle$ and $Q = \langle \mathtt{I}_0^{Q}, \dots, \mathtt{I}_{n-1}^{Q} \rangle$ are programs in the language of URM. **Concatenation** of $\mathtt{P}$ and $\mathtt{Q}$ is called the program

$$R = \langle \mathtt{I}_0^R, \dots, \mathtt{I}_{m+n-1}^R \rangle,$$

where the instructions of R are listed as follows:

$$\begin{aligned} I_0^R &= I_0^{\mathtt{P}} && \text{// a copy of } \mathtt{P} \\ &\vdots \\ I_{m-1}^R &= I_{m-1}^{\mathtt{P}} \\ I_m^R &= \mathtt{shift}(I_0^Q, \mathtt{len}(\mathtt{P})) && \text{// a copy of } Q \\ &\vdots && \text{// in which all instruction indices are shifted} \\ I_{m+n-1}^R &= \mathtt{shift}(I_{n-1}^Q, \mathtt{len}(\mathtt{P})). \end{aligned}$$

Usually we denote the concatenation of $\mathtt{P}$ and $\mathtt{Q}$ by $\mathtt{P};\mathtt{Q}$.

Let $\mathtt{Zero[}a, b\mathtt{]}$ denote the program with the following instructions:

$$\begin{aligned} &\mathtt{Zero}(a) \\ &\mathtt{Zero}(a+1) \\ &\vdots \\ &\mathtt{Zero}(b). \end{aligned}$$

Problem 16. For any two *standard* programs $\mathtt{P}$ and $\mathtt{Q}$, where $p = \mathtt{rank}(\mathtt{P})$. Show that

$$[\![\{\mathtt{P}; \mathtt{Zero[}2, p\mathtt{]}\}; \mathtt{Q}]\!] = [\![\mathtt{Q}]\!]^{(1)} \circ [\![\mathtt{P}]\!]^{(1)}.$$

2.3 Cycles in programs

Let $\mathtt{P} = \langle I^{\mathtt{P}}_0, \ldots, I^{\mathtt{P}}_{n-1}\rangle$ be a program in URM. We define the program $Q = \langle I^{Q}_0, \ldots, I^{Q}_{n+1}\rangle$, where:

$$\begin{aligned}
\mathtt{I}^Q_0 &= \mathtt{Jump}(i, j, n+2)\\
\mathtt{I}^Q_1 &= \mathtt{shift}(\mathtt{I}^{\mathtt{P}}_0, 1)\\
&\vdots\\
\mathtt{I}^Q_n &= \mathtt{shift}(\mathtt{I}^{\mathtt{P}}_{n-1}, 1)\\
\mathtt{I}^Q_{n+1} &= \mathtt{Jump}(m, m, 0).
\end{aligned}$$

We denote the program Q by "`while r[i] != r[j] do P`".

Proposition 2.1. For any standard program `P` and any state σ, we have the following:

a) if $\sigma\mathtt{[i]} \neq \sigma\mathtt{[j]}$, then

$$[\![\mathtt{while\ r[i]\ !=\ r[j]\ do\ P}]\!](\sigma) \simeq [\![\mathtt{P;}\{\mathtt{while\ r[i]\ !=\ r[j]\ do\ P}\}]\!](\sigma).$$

b) if $\sigma\mathtt{[i]} = \sigma\mathtt{[j]}$, then

$$[\![\mathtt{while\ r[i]\ !=\ r[j]\ do\ P}]\!](\sigma) = \sigma\mathtt{[1]}.$$

Example 2. The function $x+y$ is URM-computable. To see that, we define the program `P` as follows:

$$\begin{aligned}
I^{\mathtt{P}}_0 = {} & \mathtt{Jump}(1,3,4) && \text{// loop until } \mathtt{r[2]} = \mathtt{r[3]}\text{; then we are done}\\
& \mathtt{Succ}(1)\\
& \mathtt{Succ}(3)\\
& \mathtt{Jump}(1,1,0) && \text{// go to } I^{\mathtt{P}}_0
\end{aligned}$$

It is easy to see that for any two natural numbers x and y,

$$[\![\mathtt{P}]\!]^{(2)}(x,y) \simeq x+y.$$

This can be denoted by

$$[\![\mathtt{while\ r[1]} \neq \mathtt{r[3]\ do\ \{\ Succ(3);Succ(2)\ \}}]\!]^{(2)}(x,y) \simeq x+y.$$

Notice that if we consider the function of three arguments modelled by the program `P`, then we would obtain the following:

$$[\![\texttt{P}]\!]^{(3)}(x,y,z) \simeq \begin{cases} x+y-z, & \text{if } y \geq z \\ \uparrow, & \text{if } y < z. \end{cases}$$

Again, it is not hard to see that if we consider the function of one argument modelled by the program `P`, then we would obtain $[\![\texttt{P}]\!]^{(1)}(x) = x$ for all $x \in \mathbb{N}$.

Let $\texttt{P} = \langle I_0^{\texttt{P}}, \ldots, I_{n-1}^{\texttt{P}} \rangle$ be a program. We define $Q = \langle I_0^Q, \ldots, I_{n+1}^Q \rangle$, where:

$$\begin{aligned} I_0^Q &= \texttt{Jump}(1,1,n+1) && \text{// go to the last instruction} \\ I_1^Q &= \texttt{shift}(I_0^{\texttt{P}}, 1) && \\ &\vdots && \\ I_n^Q &= \texttt{shift}(I_{n-1}^{\texttt{P}}, 1) && \\ I_{n+1}^Q &= \texttt{Jump}(i,j,1) && \text{// iterate until } \texttt{r[}i\texttt{]} \neq \texttt{r[}j\texttt{]}. \end{aligned}$$

We denote the program Q by "`while r[i] == r[j] do P`".

Proposition 2.2. For any standard program P and any state σ, we have the following:

a) if $\sigma\texttt{[i]} = \sigma\texttt{[j]}$, then

$$[\![\texttt{while r[i] == r[j] do P}]\!](\sigma) \simeq [\![\texttt{P;\{while r[i] != r[j] do P\}}]\!](\sigma).$$

b) if $\sigma\texttt{[i]} \neq \sigma\texttt{[j]}$, then

$$[\![\texttt{while r[i] == r[j] do P}]\!](\sigma) \simeq \sigma\texttt{[1]}.$$

2.4 Superposition

Let $\mathtt{P} = \langle \mathtt{I}_0^{\mathtt{P}}, \dots, \mathtt{I}_{m-1}^{\mathtt{P}} \rangle$ be a program. It is useful to define a program $\mathtt{Q}$, which executes $\mathtt{P}$ over the values of registers $\mathtt{r[}\ell_1\mathtt{]}, \dots, \mathtt{r[}\ell_n\mathtt{]}$ and stores the result of the computation in $\mathtt{r[}\ell\mathtt{]}$. We usually denote this new program $\mathtt{Q}$ by

$$\mathtt{P}[\ell_1, \dots, \ell_n \mapsto \ell].$$

Let $p = \mathtt{rank}(\mathtt{P})$. We list its commands:

$$\begin{aligned}
I_0^Q &= \mathtt{Set}(\ell_1, 1) && \text{// } \mathtt{r[1]} := \mathtt{r[}\ell_1\mathtt{]} \\
&\vdots \\
I_{n-1}^Q &= \mathtt{Set}(\ell_n, n) && \text{// } \mathtt{r[}n\mathtt{]} := \mathtt{r[}\ell_n\mathtt{]} \\
I_n^Q &= \mathtt{Zero}(n+1) && \text{// hygiene} \\
I_{n+1}^Q &= \mathtt{Zero}(n+2) \\
&\vdots \\
I_{p-1}^Q &= \mathtt{Zero}(p) \\
I_p^Q &= \mathtt{shift}(I_0^{\mathtt{P}}, p) \\
&\vdots && \text{// copy of } P \\
I_{p+m-1}^Q &= \mathtt{shift}(I_{m-1}^{\mathtt{P}}, p) \\
I_{p+m}^Q &= \mathtt{Set}(1, \ell) && \text{// } \mathtt{r[}\ell\mathtt{]} := \mathtt{r[1]}
\end{aligned}$$

Let $g : \mathbb{N}^n \to \mathbb{N}$, $f_1 : \mathbb{N}^k \to \mathbb{N}, \dots, f_n : \mathbb{N}^k \to \mathbb{N}$ are (partial) functions. Define the function $h : \mathbb{N}^k \to \mathbb{N}$ as

$$h(\overline{x}) \simeq z \Leftrightarrow (\exists y_1, \dots, y_n)[f_1(\overline{x}) \simeq y_1 \,\&\, \dots \,\&\, f_n(\overline{x}) \simeq y_n \,\&\, g(y_1, \dots, y_n) \simeq z].$$

We say that h is the **superposition** of g and $f_1, \dots, f_n$. We denote

$$h = g(f_1, \dots, f_n).$$

Lemma 2.2. If $g, f_1, \dots, f_k$ are URM-computable, then $h = g(f_1, \dots, f_k)$ is URM-computable.

Proof. Let G be the program for the computable function g and let F_i be the programs for f_i, $i = 1, \ldots, k$. We need a place to store some temporary values. For this purpose, we will fix m to be a register index which is beyond any register ever touched by any of the programs G and $F_1, \ldots, F_k$. We let

Of course, all of G, $F_1, \ldots, F_k$ are in standard form.

$$m = \max\{k, n, \texttt{rank}(G), \texttt{rank}(F_1), \ldots, \texttt{rank}(F_k)\}.$$

The procedure is the following:

- We store the values of $x_1, \ldots, x_n$ into the registers $\texttt{r[}m+1\texttt{]}, \ldots, \texttt{r[}m+n\texttt{]}$ and consider these registers as read-only.

- For $i = 1, \ldots, k$, we execute the program F_i over the values stored in $\texttt{r[}m+1\texttt{]}, \ldots, \texttt{r[}m+n\texttt{]}$. and store the result in $\texttt{r[}m+n+i\texttt{]}$.

- Finally, we execute the program G over the values stored in

  $$\texttt{r[}m+n+1\texttt{]}, \ldots, \texttt{r[}m+n+k\texttt{]},$$

  and store the final result in r[1].

Then the program Q for computing the function h is the following:

```
Set$(1, m+1)$                                        // store the original values
Set$(2, m+2)$
⋮
Set$(n, m+n)$
$F_1[m+1, \ldots, m+n \to m+n+1]$;   // $\texttt{r[}m+n+1\texttt{]} := y_1 = f_1(x_1, \ldots, x_n)$
⋮
$F_k[m+1, \ldots, m+n \to m+n+k]$;   // $\texttt{r[}m+n+k\texttt{]} := y_k = f_k(x_1, \ldots, x_n)$
$G[m+n+1, \ldots, m+n+k \to 1]$                  // $\texttt{r[1]} := g(y_1, \ldots, y_k)$.
```

□

2.5 Primitive Recursion

Definition 2.1. Let f be n-ary (partial) function and g be $(n+2)$-ary (partial) function. We say that the $(n+1)$-ary (partial) function h is obtained from f and g by **primitive recursion**, if h is defined by the following scheme:

$$\left| \begin{array}{lcl} h(\overline{x}, 0) & \simeq & f(\overline{x}) \\ h(\overline{x}, y+1) & \simeq & g(\overline{x}, y, h(\overline{x}, y)). \end{array} \right.$$

Lemma 2.3. Let $f : \mathbb{N}^n \to \mathbb{N}$ and $g : \mathbb{N}^{n+2} \to \mathbb{N}$ be *URM-computable.* Then $h : \mathbb{N}^{n+1} \to \mathbb{N}$, obtained from f and g by recursion, is *URM-computable.*

Hint. The initial state is $\sigma_0 = (0, \bar{x})$. Let

$$m = \max\{n+2, \texttt{rank}(F), \texttt{rank}(G)\}.$$

We can informally describe the procedure as follows:

- We store the values of $x_1, \dots, x_n$ into the registers $\texttt{r[}m+1\texttt{]}, \dots, \texttt{r[}m+n\texttt{]}$ and consider these registers as read-only.
- We store the value of y into the register $\texttt{r[}m+n+3\texttt{]}$.
- We store the counter k in $\texttt{r[}m+n+1\texttt{]}$, and by the choice of m, its initial value is 0.
- We store in $\texttt{r[}m+n+2\texttt{]}$ consecutive values $h(\overline{x}, k)$, for $k \leq y$.
- When $\texttt{r[}m+n+2\texttt{]} = \texttt{r[}m+n+3\texttt{]}$, i.e. $k = y$, we store the value of $\texttt{r[}m+n+2\texttt{]}$ in $\texttt{r[1]}$ and exit.

We list the sequence of commands:

```
    0 : Set(1, m+1);                                  // r[m+1] := x_1
        ⋮
n − 1 : Set(n, m+n);                                  // r[m+n] := x_n
    n : Set(n+1, m+n+3);                              // r[m+n+3] := y
        F[1, 2, ..., n ↦ m+n+2];                      // r[m+n+2] := f(x̄)
    q : Jump(m+n+1, m+n+3, p);                        // if k = y then exit
        G[m+1, ..., m+n+2 ↦ m+n+2];  // r[m+n+2] := g(x̄, k, h(x̄, k))
        Succ(m+n+1);                                  // k := k+1
        Jump(1, 1, q);                                // go to I_q
    p : Set(m+n+2, 1)                                 // r[1] := h(x̄, y).
```

□

2.6 Minimisation

Definition 2.2. Let $f : \mathbb{N}^{n+1} \to \mathbb{N}$ be a (partial) function. We define the (partial) function $g : \mathbb{N}^n \to \mathbb{N}$, denoted

$$g(\overline{x}) \overset{\text{def}}{=} \mu y[f(\overline{x}, y) \simeq 0],$$

in the following way:

- $g(\overline{x}) \simeq y$, the least y for which $(\forall z \leq y)[f(\overline{x}, y) \downarrow]$ and $f(\overline{x}, y) = 0$;
- $g(\overline{x}) \uparrow$, if there is no such y.

Lemma 2.4. If the function $f(\overline{x}, y)$ is *URM-computable*, then the function

$$g(\overline{x}) = \mu y[f(\overline{x}, y) \simeq 0]$$

is also *URM-computable.*

Proof. Let $m = \max\{n+1, \mathtt{rank}(F)\}$, where F is the standard program for f.

We can describe the procedure for computing the value of $g(\overline{x})$ as follows:

- We store $x_1, \dots, x_n$ into the registers $\mathtt{r[}m+1\mathtt{]}, \dots, \mathtt{r[}m+n\mathtt{]}$ and consider these registers as read-only.
- We store the value of the counter y in $\mathtt{r[}m+n+1\mathtt{]}$. Initially y is zero, so $y = \mathtt{r[}m+n+1\mathtt{]}$, by the choice of m.
- We store in $\mathtt{r[1]}$ consecutive values $f(\overline{x}, y)$, for $y = 0, 1, \dots$
- When we first observe that $\mathtt{r[1]} = \mathtt{r[}m+n+2\mathtt{]}$, that is, for the current value of y we have $f(\overline{x}, y) = 0$, we store the value of y in $\mathtt{r[1]}$ and exit.

We list the sequence of instructions:

```
    0 : $\mathtt{Set}(1, m+1)$;                          // $\mathtt{r[}m+1\mathtt{]} := \mathtt{r[1]}$
        ⋮
$n-1$ : $\mathtt{Set}(n, m+n)$;                          // $\mathtt{r[}m+n\mathtt{]} := \mathtt{r[}n\mathtt{]}$
        $F[m+1, \dots, m+n+1 \to 1]$;                 // $\mathtt{r[1]} := f(\overline{x}, y)$
        $\mathtt{Jump}(1, m+n+2, p)$;          // if $f(\overline{x}, y) = 0$ then jump to exit
        $\mathtt{Succ}(m+n+1)$;                               // $y := y+1$
        $\mathtt{Jump}(1, 1, n)$;                     // execute $f(\overline{x}, y+1)$
    $p$ : $\mathtt{Set}(m+n+1, 1)$                          // $\mathtt{r[1]} := y$
```

□

2.7 A function which is not URM-computable

- The **busy beaver** functions is defined as

Here we follow [1, p. 83].

$$B(n) \overset{\text{def}}{=} \max\{[\![\mathtt{P}]\!]^{(1)}(0) \mid \mathtt{P} \text{ is an URM program with at most } n \text{ instructions}\}.$$

- It is easy to see that $B(1) = 1$ and $B(2) = 2$;
- Check that $B(10) \geq 39$;

Lemma 2.5. For any natural number n, there are only *finitely many functions* computed by a URM program with at most n instructions.

Hint. We use the facts:

- A program with at most n instructions uses at most $2n$ registers.
- If a URM program `P` uses $2n$ different registers, then `P` is equivalent to a program $\mathtt{P}^\star$, which uses only registers $\mathtt{r[1]}, \dots, \mathtt{r[}2n\mathtt{]}$.
- We can safely assume that we only consider standard programs `P`, i.e. the jump instructions in `P` have the form $\mathtt{Jump}(i, j, q)$, where $q \leq \mathtt{len}(\mathtt{P})$, and $1 \leq i, j \leq \mathtt{len}(\mathtt{P})$.
- Such $\mathtt{P}^\star$ chooses its instructions from a finite set with size $4n(n^2 + 2n + 1)$, since we have:

$$\begin{aligned} \mathtt{Zero}(i) &: 2n \text{ instructions} \\ \mathtt{Succ}(i) &: 2n \text{ instructions} \\ \mathtt{Set}(i, j) &: 2n * 2n \text{ instructions} \\ \mathtt{Jump}(i, j, k) &: 2n * 2n * (n + 1) \text{ instructions.} \end{aligned}$$

- Thus, the number of functions computed by URM programs with $\leq n$ instructions is bounded by $(4n(n^2 + 2n + 1))^n$, because every such program is a word of length $\leq n$ over an alphabet of size $4n(n^2 + 2n + 1)$.

□

Since the number of function computable by URM programs with $\leq n$ instructions is finite, it follows that **the busy beaver function B is total**. Let us denote by $\mathtt{P}_n$ the program with $\leq n$ instructions such that $[\![\mathtt{P}_n]\!]^{(1)}(0) = B(n)$.

Lemma 2.6. B is strictly increasing function, i.e. for every natural number n,

$$B(n) < B(n+1).$$

Hint. Clearly $B(n+1) \geq [\![P_n; \texttt{Succ}(1)]\!]^{(1)}(0)$. □

Lemma 2.7. For all $n \geq 1$, $B(n+5) \geq 2n$.

Hint. For a fixed number n, consider the program Q_n:

Here the number of instructions is important so we must explicitly list the instructions of Q_n.

$$\begin{array}{rll}
0 : & \texttt{Succ}(1) & \\
 & \vdots & \\
n-1 : & \texttt{Succ}(1) & // \; \texttt{r[1]} = n \\
n : & \texttt{Set}(2,1) & // \; \texttt{r[2]} := \texttt{r[1]} \\
n+1 : & \texttt{Jump}(2,3,n+5) & // \text{ exit if } \texttt{r[2]} = \texttt{r[3]} \\
 & \texttt{Succ}(1) & \\
 & \texttt{Succ}(3) & \\
n+4 : & \texttt{Jump}(1,1,n+1) & // \text{ goto } \texttt{I}_{n_1}
\end{array}$$

Clearly, $[\![Q_n]\!]^{(1)}(0) = 2n$ and Q_n has $n+5$ instructions. Thus, $B(n+5) \geq 2n$. □

Proposition 2.3. Every URM-computable function is dominated by a *strictly increasing* URM-computable function.

Hint. Consider a URM-computable function f. Define the function g such that

$$\left| \begin{array}{lcl} g(0) & \overset{\text{def}}{=} & f(0)+1 \\ g(n+1) & \overset{\text{def}}{=} & \max\{g(n), f(n+1)\}+1. \end{array} \right.$$

It is easy to see that g is URM-computable. □

Lemma 2.8. The busy beaver function B dominates every URM-computable function.

Hint. It is enough to consider only strictly increasing URM-computable functions g. We will show that B dominates any such g, i.e.

$$(\exists k_0)(\forall n \geq k_0)[g(n) < B(n)].$$

Let G be a URM program for g and let $k_0 = \texttt{len}(G)$. Fix $p = \texttt{rank}(P_n)$. Let G^+ be the same as G with the exception that we let G^+ work on register `r[1]` and all other register indices used in G are shifted by p positions to the right. For instance, if $I_j^G = \texttt{Set}(1, m)$ and $m > 1$, then $I_j^{G^+} = \texttt{Set}(1, m + p)$. The concatenation $P_n; G^+$ of these two programs has at most $n + k_0$ instructions. It follows that $B(n + k_0) \geq g(B(n))$.

$$\begin{aligned}
B(n + k_0 + 6) &> B(n + k_0 + 5) && \text{// } B \text{ is strictly increasing} \\
&\geq g(B(n + 5)) && \text{// as we just saw} \\
&\geq g(2n) && \text{// } B(n + 5) \geq 2n \\
&> g(n + k_0 + 6) && \text{// for } n > k_0 + 6
\end{aligned}$$

This means that $(\forall m > 2(k_0 + 6))[B(m) > g(m)]$. $\square$

Theorem 2.1 (Tibor Radó, 1962)**.** The busy beaver function B is not URM-computable.

Chapter 3

Partial recursive functions

One person's data is another person's program. Guy L. Steele,Jr.

Definition 3.1 (Kleene)**.** The **partial recursive** functions are:

1) the primitive recursive functions;

2) if f is $(n+1)$-ary partial recursive, then the n-ary function g, defined as
$$g(\overline{x}) \overset{\text{def}}{\simeq} \mu z[f(\overline{x}, z) \simeq 0],$$
is also partial recursive;

3) all partial recursive functions are obtained by rules 1) and 2).

We have already seen that the URM-computable functions are closed under superposition, the primitive recursive scheme and minimisation. It follows that all partial recursive functions are URM-computable as well.

Now we will see that we also have the converse direction.

3.1 Enumeration of URM programs

Given an instruction $\mathtt{I}$, **the code of** $\mathtt{I}$, denoted $\ulcorner \mathtt{I} \urcorner$, is the number:

π and π_3 are defined in Section 1.4.2.

$$\ulcorner \mathtt{I} \urcorner \overset{\text{def}}{=} \begin{cases} 4 * (n-1), & \text{if } \mathtt{I} = \mathtt{Zero}(n) \\ 4 * (n-1) + 1, & \text{if } \mathtt{I} = \mathtt{Succ}(n) \\ 4 * \pi(m-1, n-1) + 2, & \text{if } \mathtt{I} = \mathtt{Set}(m, n) \\ 4 * \pi_3(m-1, n-1, q) + 3, & \text{if } \mathtt{I} = \mathtt{Jump}(m, n, q) \end{cases}$$

Different instructions have different codes and every natural number is a code of some instruction. Given a program $\mathtt{P} = \langle \mathtt{I}_0^{\mathtt{P}}, \ldots, \mathtt{I}_{m-1}^{\mathtt{P}} \rangle$, **the code of** $\mathtt{P}$, denoted $\ulcorner \mathtt{P} \urcorner$, is the number

✍ Verify it!

$$\ulcorner \mathtt{P} \urcorner \overset{\text{def}}{=} \tau(\ulcorner \mathtt{I}_0^{\mathtt{P}} \urcorner, \ldots, \ulcorner \mathtt{I}_{m-1}^{\mathtt{P}} \urcorner),$$

where τ is some primitive recursive coding of finite sequences of numbers. It is easy to see that different programs have different codes and every natural number is the code of some program.

We enumerate all k-ary URM-computable functions

$$\varphi_0^{(k)}, \varphi_1^{(k)}, \ldots, \varphi_a^{(k)}, \ldots,$$

where $\varphi_a^{(k)}$ is the (partial) k-ary function, computable by the URM program with code a.

3.2 The universal function

Definition 3.2. Let us denote by $\mathcal{F}_n$ the class of all partial functions on n arguments. We say that $U(a, \overline{x})$ is *universal* for the class $\mathcal{K} \subseteq \mathcal{F}_n$ if

a) U is computable,

b) for each $f \in \mathcal{K}$, there exists $a \in \mathbb{N}$ such that $f(\overline{x}) \simeq U(a, \overline{x})$ for all $\overline{x} \in \mathbb{N}^n$, and

c) for each $a \in \mathbb{N}$, there is a function $f \in \mathcal{K}$ such that $U(a, \overline{x}) \simeq f(\overline{x})$ for all $\overline{x} \in \mathbb{N}^n$.

Example 3. A total function $p(x)$ is called *polynomial* if there exists numbers $a_0, \dots, a_n$ such that for all x, $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$. Let us denote by $\mathcal{P}$ the class of all polynomial functions. We will show that there exists a universal function U for the class $\mathcal{P}$. To see that, first notice that every polynomial function $p(x)$ is uniquely determined by the sequence $a_0, \dots, a_n$ of its coefficients. It follows that $p(x)$ is uniquely determined by the number $a = \tau(a_0, \dots, a_n)$. Denote this polynomial function as $p_a(x)$. Now it is not hard to see that $U(a, x) \overset{\text{def}}{=} p_a(x)$ is universal for the class $\mathcal{P}$.

Example 4. The class of all *total computable unary functions* is not universal.

In this section we will see that the class $\mathcal{C}_n$ of all computable functions on n arguments possess a universal function, which we will denote by $\Phi_n(a, \overline{x})$.

For every state $\sigma = (\ell, a_1, a_2, \dots, a_r, 0, 0, \dots)$, the code of σ, denoted $\ulcorner\sigma\urcorner$, is the number

We need -1 to cover the zero.

$$\ulcorner\sigma\urcorner \overset{\text{def}}{=} p_0^{\ell} \cdot p_1^{a_1} \cdot p_2^{a_2} \cdots - 1.$$

It is easy to see that every state has a different code and every natural number is a code of some state. We will use the following primitive recursive functions

$$\texttt{head}(u, n) \overset{\text{def}}{=} p_0^{(u+1)_0+1} \cdot \prod_{i=1}^{n \dot{-} 1} p_i^{(u+1)_i}$$

$$\texttt{tail}(u, n) \overset{\text{def}}{=} \prod_{i=n+1}^{\infty} p_i^{(u+1)_i} = \prod_{i=n+1}^{u+1} p_i^{(u+1)_i}.$$

Proposition 3.1. There exists a *primitive recursive* function **apply** such that for every instruction `I` and every state σ,

$[\![\mathtt{I}]\!]$ is defined in Chapter 2.

$$\mathbf{apply}(\ulcorner \mathtt{I} \urcorner, \ulcorner \sigma \urcorner) = \ulcorner [\![\mathtt{I}]\!](\sigma) \urcorner.$$

Proof. It is not difficult to see that the following function is primitive recursive.

$$\mathbf{apply}(c,u) \overset{\text{def}}{=} \begin{cases} \mathtt{head}(u,n) \cdot \boldsymbol{p_n^0} \cdot \mathtt{tail}(u,n), & \text{if } c = \ulcorner \mathtt{Zero}(n) \urcorner \\ \mathtt{head}(u,n) \cdot \boldsymbol{p_n^{(u+1)_n+1}} \cdot \mathtt{tail}(u,n), & \text{if } c = \ulcorner \mathtt{Succ}(n) \urcorner \\ \mathtt{head}(u,n) \cdot \boldsymbol{p_n^{(u+1)_m}} \cdot \mathtt{tail}(u,n), & \text{if } c = \ulcorner \mathtt{Set}(m,n) \urcorner \\ \mathtt{head}(u,0) \cdot \mathtt{tail}(u,0), & \text{if } c = \ulcorner \mathtt{Jump}(m,n,q) \urcorner \\ & \& \ (u)_n \neq (u)_m \\ \boldsymbol{p_0^q} \cdot \mathtt{tail}(u,0), & \text{if } c = \ulcorner \mathtt{Jump}(m,n,q) \urcorner \\ & \& \ (u)_n = (u)_m \end{cases}$$

□

Recall that $[\![P]\!]_s(\sigma)$ produces the program state after s steps of the execution of P over the initial state σ.

Proposition 3.2. For every number n, there exists a *primitive recursive* function **exec** such that for any state σ and number of steps s, it has the property

$$\mathbf{exec}(\ulcorner \mathtt{P} \urcorner, \ulcorner \sigma \urcorner, s) = \ulcorner [\![\mathtt{P}]\!]_s(\sigma) \urcorner.$$

Proof. Define the *primitive recursive* function **step**

$$\mathbf{step}(a,u) \overset{\text{def}}{=} \begin{cases} \mathbf{apply}(\mathtt{mem}(a,(u+1)_0),u), & \text{if } (u+1)_0 < \mathtt{len}(a) \\ u, & \text{otherwise.} \end{cases}$$

Let `P` be the program such that $a = \ulcorner \mathtt{P} \urcorner$ and σ be the state for which $u = \ulcorner \sigma \urcorner$. Then if the current state of `P` is σ, $\mathtt{mem}(a,(u+1)_0)$ gives the code of the next instruction to be executed.

$$\left| \begin{array}{lcl} \mathbf{exec}(a,u,0) & \overset{\text{def}}{=} & u \\ \mathbf{exec}(a,u,s+1) & \overset{\text{def}}{=} & \mathbf{step}(a,\mathbf{exec}(a,u,s)). \end{array} \right.$$

The function **exec** is defined by a primitive recursive scheme involving primitive recursive functions, so **exec** itself is primitive recursive. □

Theorem 3.1 (Universal function)**.** For every number n, there exists an $(n+1)$-ary *URM-computable* function Φ_n such that for every URM program with index a and every n-tuple $\bar{x}$,

$$\Phi_n(a, \bar{x}) \simeq \varphi_a^{(n)}(\bar{x}).$$

Proof. Define the *URM-computable* function `time`, which give the least number of stages for which the program with code a halts successfully on input the state σ, represented by its code u, i.e. $u = \ulcorner \sigma \urcorner$,

$$\texttt{time}(a, u) \overset{\text{def}}{\simeq} \mu s[(\mathbf{exec}(a, u, s) + 1)_0 \geq \texttt{len}(a)].$$

For any $n \geq 1$ and n-tuple $\overline{x}$, let us consider the function with the following property:

$$\texttt{init}_n(\overline{x}) = \ulcorner \underbrace{(0, x_1, x_2, \dots, x_n, 0, 0, \dots)}_{\text{initial state}} \urcorner.$$

We can define $\texttt{init}_n : \mathbb{N}^n \to \mathbb{N}$ as follows:

$$\texttt{init}_n(\overline{x}) \overset{\text{def}}{=} \prod_{i=1}^{n} p_i^{x_i} - 1.$$

Recall that the result of a successful computation is stored in the first register. Therefore, the definition of Φ_n is the following:

$$\Phi_n(a, \bar{x}) \overset{\text{def}}{\simeq} (\mathbf{exec}(a, \texttt{init}_n(\overline{x}), \texttt{time}(a, \texttt{init}_n(\overline{x}))) + 1)_1.$$

$\square$

Theorem 3.2 (Normal form, Kleene (1938))**.** For every number n, there exists an $(n+2)$-ary **primitive recursive** predicate $\texttt{T}_n$ such that for every program index a and n-tuple $\bar{x}$, we have the following:

i) $\varphi_a^{(n)}(\bar{x}) \downarrow \Leftrightarrow \ (\exists z)[\texttt{T}_n(a, \bar{x}, z) = \texttt{True}]$;

ii) $\varphi_a^{(n)}(\bar{x}) \simeq \rho(\mu z[\texttt{T}_n(a, \bar{x}, z) = \texttt{True}])$.

Proof. We define the *primitive recursive* predicate $\mathtt{res}$:

$$\mathtt{res}(a,u,s,y) \stackrel{\text{def}}{=} \begin{cases} \mathtt{True}, & (\mathbf{exec}(a,u,s)+1)_0 \geq \mathtt{len}(a) \ \&\ y = (\mathbf{exec}(a,u,s)+1)_1 \\ \mathtt{False}, & \text{otherwise} \end{cases}$$

Now we define the following predicate

$\mathtt{T}_n$ is called the Kleene predicate. We are going to use it later.

$$\boxed{\mathtt{T}_n(a,\bar{x},z) \stackrel{\text{def}}{=} \mathtt{res}(a,\mathtt{init}_n(\bar{x}),\lambda(z),\rho(z))}$$

Part i) follows from the following equivalences:

$$\begin{aligned} \varphi_a^{(n)}(\bar{x})\downarrow &\Leftrightarrow (\exists y)(\exists s)[\mathtt{res}(a,\mathtt{init}_n(\bar{x}),s,y) = \mathtt{True}] \\ &\Leftrightarrow (\exists z)[\mathtt{T}_n(a,\bar{x},z) = \mathtt{True}]. \end{aligned}$$

For part ii), we have the following:

$$\begin{aligned} \varphi_a^{(n)}(\overline{x}) \simeq y &\Leftrightarrow \Phi_n(a,\overline{x}) \simeq y \\ &\Leftrightarrow \mathtt{res}(a,\mathtt{init}_n(\overline{x}),\mathtt{time}(a,\mathtt{init}_n(\overline{x})),y) \simeq \mathtt{True} \\ &\Leftrightarrow \mathtt{T}_n(a,\overline{x},\pi(\mathtt{time}(a,\mathtt{init}_n(\overline{x})),y)) \simeq \mathtt{True} \\ &\Leftrightarrow \rho(\mu z[\mathtt{T}_n(a,\overline{x},z) = \mathtt{True}]) \simeq y. \end{aligned}$$

□

Corollary 3.1. Every URM-computable function is partial recursive.

Proof. Let us consider φ_a, the function computable by the URM program with code a. By ii) of *Theorem* 3.2, φ_a can be obtained by applying minimisation to a primitive recursive function. Thus, φ_a is a partial recursive function. □

Theorem 3.3. The class of partial recursive functions coincide with the class of URM computable functions.

3.3 The Parameters Theorem

The Parameters Theorem is one of the most important theorems in this course.

In the literature, it is usually called the S^m_n-theorem.

Lemma 3.1 (S^m_n theorem for $n, m = 1$)**.** There exists a **primitive recursive** function S^1_1, such that for every index a, and all input values x and y,

$$\varphi^{(2)}_a(x, y) \simeq \varphi^{(1)}_{S^1_1(a,x)}(y).$$

Before proceeding to the proof of the lemma, we need to do some technical preliminary work.

Proposition 3.3. The following function is *primitive recursive*:

$$\mathbf{shift}(c, s) = \begin{cases} c, & \text{if } c \text{ is arithmetical instruction} \\ \ulcorner \mathtt{Jump}(m, n, q + s) \urcorner, & \text{if } c = \ulcorner \mathtt{Jump}(m, n, q) \urcorner. \end{cases}$$

Proof. Define the primitive recursive function

$$\mathtt{step}(c, s) \overset{\text{def}}{=} 4 \cdot \pi_3(J^3_1(\mathtt{qt}(4, c)), J^3_2(\mathtt{qt}(4, c)), J^3_3(\mathtt{qt}(4, c)) + s) + 3.$$

Then it is trivial to check that:

$$\mathbf{shift}(c, s) \overset{\text{def}}{=} \begin{cases} c, & \text{if } \mathtt{rem}(4, c) \neq 3 \\ \mathtt{step}(c, s), & \text{if } \mathtt{rem}(4, c) = 3. \end{cases}$$

□

Proposition 3.4. Let a and b be the codes of the standard programs P_a and P_b. Let $Q = P_a; P_b$. Then the following function is *primitive recursive*:

$$\mathbf{instr}(a, b, i) \overset{\text{def}}{=} \begin{cases} \ulcorner \mathtt{I}^{\mathtt{Q}}_i \urcorner, & \text{if } i < \mathtt{len}(a) + \mathtt{len}(b) \\ 42, & \text{otherwise.} \end{cases}$$

Proof. The following function is clearly primitive recursive:

$$\mathbf{instr}(a, b, i) = \begin{cases} \mathtt{mem}(a, i), & \text{if } i < \mathtt{len}(a) \\ \mathbf{shift}(\mathtt{mem}(b, i - \mathtt{len}(a)), \mathtt{len}(a))), & \text{if } \mathtt{len}(a) \leq i < \mathtt{len}(a) + \mathtt{len}(b) \\ 42, & \text{otherwise} \end{cases}$$

□

Proposition 3.5. Let a and b be the codes of the programs in *standard form* $\mathtt{P}_a$ and $\mathtt{P}_b$, respectively. There exists a *primitive recursive* function **concat** such that $\mathbf{concat}(a,b)$ is the code of the program in standard form $Q = \mathtt{P}_a;\mathtt{P}_b$. In other words,

$$\mathbf{concat}(a,b) = \tau(\ulcorner \mathtt{I}_0^Q \urcorner, \dots, \ulcorner \mathtt{I}^Q_{\mathtt{len}(a)+\mathtt{len}(b)-1} \urcorner).$$

Proof. We know that since **instr** is primitive recursive, the *history* of **instr** is also primitive recursive. By the definition of Q as the concatenation of $\mathtt{P}_a$ and $\mathtt{P}_b$, if q is the code of Q, then $\mathtt{len}(q) = \mathtt{len}(a) + \mathtt{len}(b)$. Then we can compute the code q of Q in the following way:

Recall *Lemma* 1.1.

$$\begin{aligned} q &= \ulcorner Q \urcorner \\ &= \tau(\ulcorner \mathtt{I}_0^Q \urcorner, \dots, \ulcorner \mathtt{I}^Q_{\mathtt{len}(q)-1} \urcorner) \\ &= \tau(\mathbf{instr}(a,b,0), \mathbf{instr}(a,b,1), \dots, \mathbf{instr}(a,b,\mathtt{len}(a)+\mathtt{len}(b)-1)) \\ &= H_{\mathbf{instr}}(a,b,\mathtt{len}(a)+\mathtt{len}(b)-1). \end{aligned}$$

This shows that we can take

$$\mathbf{concat}(a,b) = H_{\mathbf{instr}}(a,b,\mathtt{len}(a)+\mathtt{len}(b)-1).$$

By *Lemma* 1.1, since $H_{\mathbf{instr}}$ is primitive recursive, so is **concat**. □

For a given number x, consider the program $\mathtt{R}_x = \langle \mathtt{I}_1, \dots, \mathtt{I}_{x+2} \rangle$, which moves the value of the first register to the second and sets the value of `r[1]` to the number x. Here is the instruction list of the program $\mathtt{R}_x$:

The program $\mathtt{R}_x$ is in standard form.

```
    1 : Set(1,2)        // r[2] := r[1]
    2 : Zero(1)         // r[1] := 0
    3 : Succ(1)         // r[1]++
          ⋮
x + 2 : Succ(1)         // r[1] == x
```

Proposition 3.6. There exists a *primitive recursive* function **regcpy** such that $\mathbf{regcpy}(x) = \ulcorner \mathtt{R}_x \urcorner$.

Proof. Firstly, we have the *primitive recursive* function **inc**, where

$$\left|\begin{array}{lcl} \mathbf{inc}(0) & \overset{\text{def}}{=} & \pi(\ulcorner \mathtt{Set}(1,2) \urcorner, \ulcorner \mathtt{Zero}(1) \urcorner) \\ \mathbf{inc}(x+1) & \overset{\text{def}}{=} & \pi(\mathbf{inc}(x), \ulcorner \mathtt{Succ}(1) \urcorner) \end{array}\right.$$

Then we define

$$\mathbf{regcpy}(x) \overset{\text{def}}{=} \pi(x+1, \mathbf{inc}(x)).$$

$\square$

We have that $\mathbf{regcpy}(x) = \tau(\ulcorner \mathtt{I}_1 \urcorner, \ldots, \ulcorner \mathtt{I}_{x+2} \urcorner)$, i.e. $\mathbf{regcpy}(x)$ is a code for the program R_x. Then for any program P, we have

$$[\![R_x; \mathtt{P}]\!](y) \simeq [\![\mathtt{P}]\!](x, y).$$

Now we have all ingredients to finish the proof of the Parameters Theorem for the case $n, m = 1$. For all numbers a and x, we define

$$S^1_1(a, x) \overset{\text{def}}{=} \mathbf{concat}(\mathbf{regcpy}(x), a).$$

It is easy to verify that for any URM program $\mathtt{P}$ and natural number x,

$$S^1_1(\ulcorner \mathtt{P} \urcorner, x) = \ulcorner \mathtt{R}_x; \mathtt{P} \urcorner.$$

Since **concat** and **regcpy** are primitive recursive, and primitive recursion is preserved under superposition, S^1_1 is also primitive recursive.

Remark. The $S^1_1(a, x)$ function is strictly monotonically increasing on the second argument x. Later we will use this fact in a few problems.

Now we are ready to define the primitive recursive function S^m_n for all numbers m and n.

Theorem 3.4 (The Parameters Theorem or The S^m_n theorem)**.** Prove that there exists a **primitive recursive** function S^m_n such that for every index a and every m-tuple $\bar{x}$ and n-tuple $\bar{y}$,

$$\varphi^{(m+n)}_a(\bar{x}, \bar{y}) \simeq \varphi^{(n)}_{S^m_n(a, \bar{x})}(\bar{y}).$$

Proof. We divide the proof in two steps.

(1) Prove the theorem for every n, i.e. for every index a, every input value x and n-tuple $\bar{y}$, ✍ Homework!

$$\varphi^{(1+n)}_a(x, \bar{y}) \simeq \varphi^{(n)}_{S^1_n(a, x)}(\bar{y}).$$

We build S^1_n is a way very similar to the way we built S^1_1.

(2) Now we proceed by induction on m.

- For $m = 1$, we already have S^1_n, for every n.
- For $m = k + 1$, we show the existence of S^{k+1}_n, for every n. Let $x, \bar{x}$ be any $(1 + k)$-tuple and $\bar{y}$ be any n-tuple. Then we have the following:

$$\begin{aligned} \varphi^{(1+k+n)}_a(x, \bar{x}, \bar{y}) &\simeq \varphi^{(k+n)}_{S^1_{k+n}(a,x)}(\bar{x}, \bar{y}) && \text{// by (1)} \\ &\simeq \varphi^{(n)}_{S^k_n(S^1_{k+n}(a,x),\bar{x})}(\bar{y}) && \text{// by I.H. for (2)} \end{aligned}$$

  Therefore, we define

$$S^{k+1}_n(a, x, \bar{x}) \overset{\text{def}}{=} S^k_n(S^1_{k+n}(a, x), \bar{x}).$$

□

3.4 Applications

Before moving on, let us explain again when we write $\varphi^{(n)}_{h(x)}(\bar{y})$, where h is some computable function, we mean that on input x and $\bar{y}$, first we compute the value $a = h(x)$ and then we compute $\varphi^{(n)}_a$ on input $\bar{y}$. More formally,

$$\varphi^{(n)}_{h(x)}(\bar{y}) \overset{\text{def}}{\simeq} \Phi_n(h(x), \bar{y}).$$

Let us introduce the notation $W_a \overset{\text{def}}{=} \texttt{dom}(\varphi^{(1)}_a)$.

Problem 17. Show that there exists a total computable function h such that for all natural numbers a,

$$W_{h(a)} = \{a\}.$$

Hint. Consider the computable function

$$f(x,y) \overset{\text{def}}{\simeq} \begin{cases} \texttt{True}, & x = y \\ \uparrow, & \text{otherwise.} \end{cases}$$

Since f is computable, then there exists an index e such that $f = \varphi^{(2)}_e$. Then let $h(x) = S^1_1(e, x)$. Now we have the following chain of equivalences:

$$\begin{aligned} x \in W_{h(a)} &\Leftrightarrow \varphi_{h(a)}(x)\downarrow \\ &\Leftrightarrow \varphi_e(a,x)\downarrow \\ &\Leftrightarrow f(a,x)\downarrow \\ &\Leftrightarrow x = a. \end{aligned}$$

We conclude that $W_{h(a)} = \{a\}$. □

Problem 18. Show that there exist primitive recursive functions f and g such that:

- $W_{f(a,b)} = W_a \cup W_b$;

- $W_{g(a,b)} = W_a \cap W_b$;

Proof. First, consider the following computable function:

$$F(a,b,x) \overset{\text{def}}{\simeq} (\mu z)[\mathtt{T}_1(a,x,\lambda(z)) = \mathtt{True} \vee \mathtt{T}_1(a,x,\rho(z)) = \mathtt{True}].$$

By the Parameters Theorem, there exists a primitive recursive function f such that for all x,

$$\varphi^{(1)}_{f(a,b)}(x) \simeq F(a,b,x).$$

Not it is easy to see that $W_{f(a,b)} = W_a \cup W_b$.

Second, consider the following computable function:

$$G(a,b,x) \overset{\text{def}}{\simeq} (\mu z)[\mathtt{T}_1(a,x,\lambda(z)) = \mathtt{True} \wedge \mathtt{T}_1(a,x,\rho(z)) = \mathtt{True}].$$

By the Parameters Theorem, there exists a primitive recursive function g such that for all x,

$$\varphi^{(1)}_{g(a,b)}(x) \simeq G(a,b,x).$$

Not it is easy to see that $W_{g(a,b)} = W_a \cap W_b$. □

Problem 19. Show that there does *not* exist a computable function h such that for all indices a,

$$W_{h(a)} = \mathbb{N} \setminus W_a.$$

Proof. Assume that such a computable function h exists. Then there exists a computable function f such that

$$f(x) \simeq \varphi^{(1)}_{h(x)}(x).$$

Since f is computable, then $f = \varphi^{(1)}_e$ for some index e. Then we have the following chains of implications:

$$\begin{array}{lllllllll} f(e)\downarrow & \Longrightarrow & \varphi_e(e)\downarrow & \Longrightarrow & e \in W_e & \Longrightarrow & e \notin W_{h(e)} & \Longrightarrow & f(e)\uparrow \\ f(e)\uparrow & \Longrightarrow & \varphi_e(e)\uparrow & \Longrightarrow & e \notin W_e & \Longrightarrow & e \in W_{h(e)} & \Longrightarrow & f(e)\downarrow. \end{array}$$

In both cases, we reach a contradiction. □

Problem 20. Show that there exists a primitive recursive function h such that for all indices a,

$$W_{h(a)} = \mathbb{N} \setminus \{0,1,\dots,a\}.$$

Theorem 3.5 (Recursion Theorem (Kleene))**.** Let h be any $(n+1)$-ary computable function. There exists an index e such that for any n-tuple $\bar{x}$,

$$h(e,\bar{x}) \simeq \varphi_e^{(n)}(\bar{x}).$$

[3, p. 78]

Proof. Consider the computable function

Later we will give another proof of this result.

$$g(z,\bar{x}) \overset{\text{def}}{\simeq} h(S_n^1(z,z),\bar{x}).$$

Since g is computable, there exists an index a such that $g = \varphi_a^{(n+1)}$. Then by the Parameters Theorem,

$$g(a,\bar{x}) \simeq \varphi_a^{(n+1)}(a,\bar{x}) \simeq \varphi_{S_n^1(a,a)}^{(n)}(\bar{x}).$$

We let $e = S_n^1(a,a)$. Combining everything we know so far, we get:

$$\begin{aligned}
h(e,\bar{x}) &\simeq h(S_n^1(a,a),\bar{x}) && /\!/ \ e = S_n^1(a,a)\\
&\simeq g(a,\bar{x}) && /\!/ \text{ by def. of } g\\
&\simeq \varphi_a^{(n+1)}(a,\bar{x}) && /\!/ \ g = \varphi_a\\
&\simeq \varphi_{S_n^1(a,a)}^{(n)}(\bar{x}) && /\!/ \text{ by the Parameters Theorem}\\
&\simeq \varphi_e^{(n)}(\bar{x}) && /\!/ \text{ since } e = S_n^1(a,a).
\end{aligned}$$

□

Corollary 3.2. There exists an index e such that

$$(\forall x)[e \simeq \varphi_e(x)].$$

Proof. Consider the computable function $h(z,x) \overset{\text{def}}{=} z$. There exists an index e such that for all x, $h(e,x) \simeq \varphi_e(x)$. Therefore,

$$(\forall x)[e \simeq h(e,x) \simeq \varphi_e(x)].$$

□

We know that the Ackermann function is not primitive recursive, but we don't know yet that it is partial recursive, or equivalently, computable.

One way to see that the Ackermann function ψ is computable is by writing a program in the language of URM which computes ψ. Here we present another proof based on the results of this chapter.

Problem 21. The Ackermann function ψ is computable.

Proof. First let us consider the function Ψ, where

$$\Psi(a,x,y) \simeq \begin{cases} y+1, & \text{if } x=0 \\ \Phi_2(a, x \dot{-} 1, 1), & \text{if } x>0 \ \& \ y=0 \\ \Phi_2(a, x \dot{-} 1, \Phi_2(a,x,y \dot{-} 1)), & \text{if } x>0 \ \& \ y>0. \end{cases}$$

By the Recursion Theorem, there exists a program index e such that for every x, y,

$$\Psi(e,x,y) \simeq \varphi_e^{(2)}(x,y).$$

Thus, when we fix the index e, we obtain the following:

$$\begin{aligned} \varphi_e^{(2)}(x,y) &\simeq \begin{cases} y+1, & \text{if } x=0 \\ \Phi_2(e, x \dot{-} 1, 1), & \text{if } x>0 \ \& \ y=0 \\ \Phi_2(e, x \dot{-} 1, \Phi_2(e,x,y \dot{-} 1)), & \text{if } x>0 \ \& \ y>0. \end{cases} \\ &\simeq \begin{cases} y+1, & \text{if } x=0 \\ \varphi_e^{(2)}(x \dot{-} 1, 1), & \text{if } x>0 \ \& \ y=0 \\ \varphi_e^{(2)}(x \dot{-} 1, \varphi_e^{(2)}(x,y \dot{-} 1)), & \text{if } x>0 \ \& \ y>0. \end{cases} \end{aligned}$$

Now it is straightforward to observe that $\varphi_e^{(2)} = \psi$. $\square$

Proposition 3.7. The problem "φ_e is total" is undecidable. More formally, there is no total computable function g such that

$$g(e) = \begin{cases} \texttt{True}, & \text{if } \varphi_e \text{ is total} \\ \texttt{False}, & \text{if } \varphi_e \text{ is not total.} \end{cases}$$

Proof. Assume that such total computable function g exists. Diagonalize over all total computable functions. To do that, consider the total computable function h defined in the following way:

Notice that in this proof we only use the existence of universal function.

$$h(e) \overset{\text{def}}{=} \begin{cases} \varphi_e^{(1)}(e)+1, & \text{if } g(e) = \texttt{True} \\ 42, & \text{if } g(e) = \texttt{False}. \end{cases}$$

Now, fix an index a such that $\varphi_a^{(1)} = h$. Clearly, $\varphi_a^{(1)}$ is a total function, and hence $g(a) = \texttt{True}$. It follows that $h(e) = \varphi_e^{(1)}(e)+1$, but $\varphi_e^{(1)}(e) \neq \varphi_e^{(1)}(e)+1$. We reach a contradiction. $\square$

Proof. Assume that f is computable. Consider the computable function g, where

$$\left| \begin{array}{lcl} g(0) & \overset{\text{def}}{=} & (\mu y)[f(y) = \texttt{True}] \\ g(x+1) & \overset{\text{def}}{=} & (\mu y)[f(y) = \texttt{True} \ \& \ y > g(x)]. \end{array} \right.$$

It is clear that g enumerates all indices of total computable functions. Consider the total computable function h, where

$$h(x) = \varphi_{g(x)}^{(1)}(x) + 1.$$

Since h is total computable and g enumerates all total computable functions, there exists an index e such that $h = \varphi_{g(e)}^{(1)}$. Then

$$\varphi_{g(e)}^{(1)}(e) = h(e) = \varphi_{g(e)}^{(1)}(e) + 1.$$

We reach a contradiction. $\square$

This proof presents a more powerful proof method. We will use it later.

Proof. Assume that f is total computable. Consider the computable function

$$g(x,y) \overset{\text{def}}{\simeq} \begin{cases} 42, & \text{if } f(x) = \texttt{False} \\ \uparrow, & \text{if } f(x) = \texttt{True}. \end{cases}$$

By the Recursion Theorem, there exists an index e such that for all y,

$$\varphi_e^{(1)}(y) \simeq g(e,y).$$

Then we have the following chains of implications:

$$\begin{array}{l} \varphi_e^{(1)} \text{ is total} \implies f(e) = \texttt{True} \implies \varphi_e^{(1)} \text{ is not total} \\ \varphi_e^{(1)} \text{ is not total} \implies f(e) = \texttt{False} \implies \varphi_e^{(1)} \text{ is total.} \end{array}$$

We reach a contradiction. $\square$

Fix an arbitrary computable $(n+1)$-ary function f and consider the equiation

$$f(u,\bar{x}) \simeq \Phi_{n+1}(u,\bar{x}).$$

The Recursion Theorem says that this equation has a solution in the sense that there exists an index e such that

$$f(e,\bar{x}) \simeq \varphi_e^{(n)}(\bar{x}).$$

It is natural to ask whether it is possible to generalize the Recursion Theorem to solve multiple equations at the same time.

Theorem 3.6 (Smullyan)**.** Let f and g be arbitrary $(n+2)$-ary computable functions. There exist indices e_1 and e_2 such that:

[4, p. 111, Problem 17], [8, p. 190]. We will give a second proof later.

$$\left|\begin{array}{lcl} f(e_1, e_2, \bar{x}) & \simeq & \varphi^{(n)}_{e_1}(\bar{x}) \\ g(e_1, e_2, \bar{x}) & \simeq & \varphi^{(n)}_{e_2}(\bar{x}). \end{array}\right.$$

Proof. We shall used the primitive recursive coding triple $\langle \pi, \lambda, \rho \rangle$ from *Section* 1.4.2. By an easy application of the Parameters Theorem, there exist total computable functions ℓ and r such that

$$\left|\begin{array}{lcl} \lambda(\Phi_n(a, \bar{x})) & \simeq & \varphi^{(n)}_{\ell(a)}(\bar{x}) \\ \rho(\Phi_n(a, \bar{x})) & \simeq & \varphi^{(n)}_{r(a)}(\bar{x}) \end{array}\right.$$

Consider the computable function Θ, where

$$\Theta(z, \bar{x}) \overset{\text{def}}{\simeq} \pi(f(\ell(z), r(z), \bar{x}), g(\ell(z), r(z), \bar{x})).$$

By the Recursion Theorem, we know that there exists an index e such that

$$\Theta(e, \bar{x}) \simeq \varphi^{(n)}_e(\bar{x}).$$

Then for this special index e,

$$\begin{array}{rll} f(\ell(e), r(e), \bar{x}) & \simeq \lambda(\Theta(e, \bar{x})) & \text{// by def. of } \Theta \\ & \simeq \lambda(\varphi^{(n)}_e(\bar{x})) & \text{// since } \Theta(e, \bar{x}) \simeq \varphi^{(n)}_e(\bar{x}) \\ & \simeq \lambda(\Phi_n(e, \bar{x})) & \text{// Universal function} \\ & \simeq \varphi^{(n)}_{\ell(e)}(\bar{x}) & \text{// by def. of } \lambda \end{array}$$

In a similar way we prove that

$$g(\ell(e), r(e), \bar{x}) \simeq \varphi^{(n)}_{r(e)}(\bar{x}).$$

We conclude that we can take $e_1 = \ell(e)$ and $e_2 = r(e)$. □

3.5 The fixed point theorem

Recall the notation that $\varphi^{(n)}_{h(x)}(\bar{y}) \overset{\text{def}}{\simeq} \Phi_n(h(x), \bar{y})$.

Proposition 3.8. For *any* computable function h and any n, there exists a *total* computable function η such that

$$\varphi^{(n)}_{h(x)} = \varphi^{(n)}_{\eta(x)}.$$

Proof. We have the following chain of equalities:

$$\begin{aligned}
\varphi^{(n)}_{h(x)}(\bar{y}) &\simeq \Phi_n(h(x), \bar{y}) \\
&\overset{\text{def}}{\simeq} g(x, \bar{y}) && \text{// } g \text{ is computable} \\
&\simeq \varphi^{(n)}_{\eta(x)}(\bar{y}). && \text{// by the Parameters Theorem}
\end{aligned}$$

□

This result may seem surprising at first, because it implies that there exists a *total* computable function ε such that for all x,

$$\varphi^{(n)}_{\emptyset^{(1)}(x)} = \varphi^{(n)}_{\varepsilon(x)}.$$

Theorem 3.7 (Fixed point theorem)**.** Let h be a total computable function. There exists an index a such that:

$$\varphi^{(n)}_a = \varphi^{(n)}_{h(a)}.$$

Also known as the second recursion theorem of Kleene.

Proof. Consider the *computable* function ϕ, where

$$\phi(e, \bar{x}) \overset{\text{def}}{\simeq} \Phi_n(h(e), \bar{x}).$$

Since ϕ is computable, by the Recursion Theorem, there exists an index a such that for all n-tuples $\bar{x}$,

$$\phi(a, \bar{x}) \simeq \varphi^{(n)}_a(\bar{x}).$$

Then, combining all of this, we get the following chain of equalities:

$$\varphi^{(n)}_a(\bar{x}) \simeq \phi(a, \bar{x}) \simeq \Phi_n(h(a), \bar{x}) \simeq \varphi^{(n)}_{h(a)}(\bar{x}).$$

This is a short proof using the Recursion Theorem. We will later adapt this proof to get a uniform version of this theorem

□

Recall that the first proof of the Fixed Point Theorem relied on the Recursion Theorem. Now we give another proof of the Recursion Theoremwhich is based on the Fixed Point Theorem.

Corollary 3.3. For every computable $F(y, \bar{x})$, there exists an index e such that for every n-tuple $\bar{x}$,

$$F(e, \bar{x}) \simeq \varphi_e^{(n)}(\bar{x}).$$

Proof.

$$\begin{aligned} F(y, \bar{x}) &\simeq \varphi_a(y, \bar{x}) && \text{// since } F \text{ is computable} \\ &\simeq \varphi_{S_n^1(a,y)}(\bar{x}) && \text{// by the Parameters Theorem} \\ &\simeq \varphi_{f(y)}(\bar{x}). && \text{// } f(y) \overset{\text{def}}{=} S_n^1(a, y) \end{aligned}$$

Then, by the Fixed Point Theorem, there exists an index e such that for every n-tuple $\bar{x}$,

$$F(e, \bar{x}) \simeq \varphi_{f(e)}^{(n)}(\bar{x}) \simeq \varphi_e^{(n)}(\bar{x}).$$

□

The fixed point theorem says that every unary total computable function f possess a fixed point. It turns out that f has infinitely many fixed points.

Proposition 3.9. Show that for every total computable f and any n, there exist *infinitely many* e, such that

$$\varphi_e^{(n)} = \varphi_{f(e)}^{(n)}.$$

Hint. We will show that for every k, there is $e > k$ such that $\varphi_e = \varphi_{f(e)}$. Fix k. Choose e_0 such that $\varphi_{e_0} \notin \{\varphi_0, \varphi_1, \ldots, \varphi_k\}$. Define the total computable function

$$g_k(x) = \begin{cases} e_0, & x \leq k \\ f(x), & x > k. \end{cases}$$

There is a number e such that $\varphi_e^{(n)} = \varphi_{g_k(e)}^{(n)}$. By the choice of g_k, $e > k$, and hence $\varphi_e^{(n)} = \varphi_{g_k(e)}^{(n)} = \varphi_{f(e)}^{(n)}$. □

Problem 22. Show that there is an index e such that

$$\varphi_e^{(1)} = \varphi_{e+1}^{(1)}.$$

Recall that we introduced the notation $W_e \overset{\text{def}}{=} \mathtt{dom}(\varphi_e^{(1)})$.

Problem 23. Show that there is an index e such that

$$W_e = \{e\}.$$

Hint. We already know that there exists a total computable function h such that

$$W_{h(a)} = \{a\}.$$

Apply the Fixed Point Theorem to h. □

Problem 24. Show that there is an index e such that

$$W_e = \{0, 1, \dots, e\}.$$

Since this is one of our most important results, we will give yet another proof of the Fixed Point Theorem.

Problem 25. Let us consider the matrix M consisting of unary computable functions:

$$\begin{array}{llllll}
M_0: & \varphi_{\varphi_0(0)} & \varphi_{\varphi_0(1)} & \cdots & \varphi_{\varphi_0(n)} & \cdots \\
M_1: & \varphi_{\varphi_1(0)} & \varphi_{\varphi_1(1)} & \cdots & \varphi_{\varphi_1(n)} & \cdots \\
\vdots & & & \ddots & & \\
M_n: & \varphi_{\varphi_n(0)} & \varphi_{\varphi_n(1)} & \cdots & \varphi_{\varphi_n(n)} & \cdots \\
\vdots & \vdots & \vdots & & \vdots & \ddots
\end{array}$$

Prove the following:

1) the diagonal $D = \{\varphi_{\varphi_i(i)} \mid i \in \mathbb{N}\}$ coincides with some row M_n in the matrix, where φ_n is total;

2) if f is *total computable*, the sequence $D^f = \{\varphi_{f(\varphi_i(i))}\}_{i\in\mathbb{N}}$ is also a row in the matrix, say M_n, where φ_n is total;

3) for every *total computable* f, there exists an index e such that $\varphi_{f(e)} = \varphi_e$.

3) is Fixed Point Theorem.

Proof. For 1), consider a *computable* function Θ such that for every i and x,

$$\Theta(i,x) \simeq \Phi_1(\Phi_1(i,i),x) \simeq \varphi_{\varphi_i(i)}(x).$$

Recall that Φ_1 is the universal function, see *Theorem* 3.1.

We have an index e for Θ, i.e. $\Theta = \varphi_e^{(1)}$. By Parameters Theorem, for every i and x,

$$\Theta(i,x) \simeq \varphi_{S_1^1(e,i)}(x).$$

For the fixed e, let n be an index such that $\varphi_n(i) = S_1^1(e,i)$. We combine everything:

$$\begin{aligned}
\varphi_{\varphi_i(i)}(x) &\simeq \Phi_1(\Phi_1(i,i),x) && \text{// Universal function}\\
&\simeq \Theta(i,x) && \text{// by def.}\\
&\simeq \varphi_e(i,x) && \text{// } \Theta \text{ is computable}\\
&\simeq \varphi_{S_1^1(e,i)}(x) && \text{// by the Parameters Theorem}\\
&\simeq \varphi_{\varphi_n(i)}(x).
\end{aligned}$$

We conclude that D coincides with M_n.

Now we proceed with 2). Its proof is similar to that of 1). Here consider the computable function Ψ such that for every i and x,

$$\Psi(i,x) \simeq \Phi_1(f(\Phi_1(i,i)),x) \simeq \varphi_{f(\varphi_i(i))}(x).$$

$$\begin{aligned}
\varphi_{f(\varphi_i(i))}(x) &\simeq \Phi_1(f(\Phi_1(i,i)),x) && \text{// Universal function}\\
&\simeq \Psi(i,x) && \text{// by def.}\\
&\simeq \varphi_e(i,x) && \text{// } \Psi \text{ is computable}\\
&\simeq \varphi_{S_1^1(e,i)}(x) && \text{// by Parameters Theorem}\\
&\simeq \varphi_{\varphi_k(i)}(x) && \text{// for fixed } e,\ S_1^1(e,x) \simeq \varphi_k(x)\ .
\end{aligned}$$

We conclude that D^f coincides with M_k.

We turn our attention to 3). By 2), for the given *total computable* f, D^f coincides with M_k, for some k such that φ_k is total.

$$\varphi_{f(\varphi_i(i))} = \varphi_{\varphi_k(i)}.$$

Let us consider the number $e = \varphi_k(k)$, which exists, because φ_k is total. Then we have the following:

$$\begin{aligned}
\varphi_{f(e)} &= \varphi_{f(\varphi_k(k))}\\
&= \varphi_{\varphi_k(k)}\\
&= \varphi_e.
\end{aligned}$$

$\square$

The benefit of the second proof of the Fixed Point Theorem is that it can be easily generalised to get a version *with parameters*. Moreorver, we can obtain a *uniform* version of the Fixed Point Theorem. By uniformity here we mean that we can find a fixed point by computable means.

Theorem 3.8 (Uniform version)**.** There exists a total computable function η such that the following implication holds:

$$(\forall a)[\varphi_a \text{ is total} \implies \varphi_{\eta(a)} = \varphi_{\varphi_a(\eta(a))}].$$

Proof. Consider the sequence

$$D^a = \{\varphi_{\varphi_a(\varphi_i(i))}\}_{i\in\mathbb{N}}.$$

We will show that there exists a total computable h such that for any a,

$$D^a = M_{h(a)}.$$

This follows easily by a slight modification of the proof of Fixed Point Theorem.

$$\begin{aligned}
\varphi_{\varphi_a(\varphi_i(i))}(x) &\simeq \Phi_1(\Phi_1(a,(\Phi_1(i,i))),x) && \text{// universal function}\\
&\simeq \varphi_e^{(3)}(a,i,x) && \text{// for some index } e\\
&\simeq \varphi^{(1)}_{S^2_1(e,a,i)}(x) && \text{// by the Parameters Theorem}\\
&\simeq \varphi^{(1)}_{H(a,i)}(x) && \text{// let } H(a,i) \overset{\text{def}}{=} S^2_1(e,a,i)\\
&\simeq \varphi_{\varphi_{h(a)}(i)}(x) && \text{// by the Parameters Theorem.}
\end{aligned}$$

Since H is total, for $\eta(a) \overset{\text{def}}{=} \varphi_{h(a)}(h(a))$ we will obtain

$$\varphi_{\varphi_a(\eta(a))} = \varphi_{\eta(a)}.$$

$\square$

Theorem 3.9 (version with parameters)**.** If f is *total computable*, then there exists a *total computable* function η such that

$$(\forall y)[\varphi^{(n)}_{\eta(y)} = \varphi^{(n)}_{f(\eta(y),y)}].$$

Moreover, η can be taken to be one-to-one.

Proof. Here, for a computable function f, for any y, consider the sequence [10, p. 37] [2, p. 210]

$$D^{f,y} = \{\varphi_{f(\varphi_i(i),y)}\}_{i\in\mathbb{N}}.$$

We will show that there is a total computable function η such that

$$D^{f,y} = M_{h(y)}.$$

Then we let $\eta(y) = \varphi_{h(y)}(h(y))$ to obtain

$$\varphi_{f(\eta(y),y)} = \varphi_{\eta(y)}.$$

Now, for any i, y, and x,

$$\begin{aligned}
\varphi_{f(\varphi_i(i),y)}(x) &\simeq \Phi_1(f(\varphi_i(i),y),x) \\
&\simeq \Phi_1(\varphi_e(y,i),x) && \text{// for some index } e \\
&\simeq \Phi_1(\varphi_{S_1^1(e,y)}(i),x) \\
&\simeq \Phi_1(\varphi_{h(y)}(i),x) && \text{// let } h(y) = S_1^1(e,y) \\
&\simeq \varphi_{\varphi_{h(y)}(i)}(x).
\end{aligned}$$

It follows that $D^{f,y} = M_{h(y)}$. □

Corollary 3.4 (Smullyan)**.** Let f and g be $(n+2)$-ary computable functions. There exist indices e_1 and e_2 such that: We have another proof - see *Theorem* 3.6

$$\left|\begin{array}{lcl} f(e_1,e_2,\bar{x}) & \simeq & \varphi^{(n)}_{e_1}(\bar{x}) \\ g(e_1,e_2,\bar{x}) & \simeq & \varphi^{(n)}_{e_2}(\bar{x}). \end{array}\right.$$

Proof. By the Parameters Theorem, there exists a primitive recursive function S such that

$$f(z,y,\bar{x}) \simeq \varphi^{(n)}_{S(z,y)}(\bar{x}).$$

By the Parametrized fixed point theorem, there exits a computable function η such that for every y,

$$\varphi^{(n)}_{\eta(y)}(\bar{x}) \simeq \varphi^{(n)}_{S(\eta(y),y)}(\bar{x}) \simeq f(\eta(y),y,\bar{x}).$$

Now consider the computable function

$$\phi(y,\bar{x}) \simeq g(\eta(y),y,\bar{x}).$$

By the Recursion Theorem, there exists an index e such that

$$\varphi^{(n)}_e(\bar{x}) \simeq \phi(e,\bar{x}) \simeq g(\eta(e),e,\bar{x}).$$

In the end, let $e_2 = e$ and $e_1 = \eta(e)$. □

Problem 26. For two total computable functions f and g, prove that there exist indices a and b such that:

$$\left|\begin{array}{lcl} \varphi^{(n)}_{f(a,b)} & = & \varphi^{(n)}_{a} \\ \varphi^{(n)}_{g(a,b)} & = & \varphi^{(n)}_{b}. \end{array}\right.$$

Proof. By the Fixed point with parameters theorem, there exists a computable η such that for every y,

$$\varphi_{f(\eta(y),y)} = \varphi_{\eta(y)}.$$

Let $h(y) = g(\eta(y), y)$. By Fixed Point Theorem, there exists an index e such that

$$\varphi_e = \varphi_{h(e)} = \varphi_{g(\eta(e),e)}.$$

Let $b = e$ and $a = \eta(e)$. $\square$

Problem 27. Prove that for any arity n, there exists a total computable function h such that for every natural number x

$$\varphi^{(n)}_{h(x)} = \varphi^{(n)}_{\varphi_{h(x)}(x)}$$

Proof. Let us consider the computable function

$$\phi(z, y, \bar{x}) \simeq \Phi_n(\Phi_1(z, y), \bar{x}) \simeq \varphi^{(n)}_{\varphi_z(y)}(\bar{x}).$$

By the Parameters Theorem, there is a total computable S such that

We do not need the fact that S is primitive recursive.

$$\phi(z, y, \bar{x}) \simeq \varphi^{(n)}_{S(z,y)}(\bar{x}).$$

By the Fixed point with parameters theorem, there is a total computable η such that

$$\varphi_{\eta(y)} = \varphi_{S(\eta(y),y)}.$$

In the end, for every y and $\bar{x}$,

$$\begin{aligned} \varphi^{(n)}_{\eta(y)}(\bar{x}) &\simeq \varphi^{(n)}_{S(\eta(y),y)}(\bar{x}) \\ &\simeq \phi(\eta(y), y, \bar{x}) \\ &\simeq \varphi^{(n)}_{\varphi_{\eta(y)}(y)}(\bar{x}). \end{aligned}$$

$\square$

3.6 Problems

Problem 28. Show that there exists a partial recursive function f, which cannot be extended to a total recursive function g.

Hint. Consider the partial recursive function

$$f(x) \simeq \Phi_1(x, x) + 1.$$

$\square$

Let $\mathcal{K}$ be a class of n-ary functions. We say that the $(n+1)$-ary function U is universal for $\mathcal{K}$ if we have the following:

1) For any $f \in \mathcal{K}$, there is at least one number e such that $U(e, \bar{x}) \simeq f(\bar{x})$ for any $\bar{x}$.

2) For any number e, the function $\lambda\bar{x}.U(e, \bar{x}) \in \mathcal{K}$.

Problem 29. Prove that the class of n-ary total computable functions does not possess a computable universal function.

Hint. Assume that U is one such computable universal function. Consider the total computable function

$$f(x_1, \dots, x_n) \overset{\text{def}}{=} U(x_1, x_1, \dots, x_n) + 1.$$

It follows that there is some index e such that $f(\bar{x}) = U(e, \bar{x})$. But then

$$\begin{aligned} f(e, x_2, \dots, x_n) &= U(e, e, x_2, \dots, x_n) + 1 \\ &= U(e, e, x_2, \dots, x_n). \end{aligned}$$

We reach a contradiction. $\square$

Problem 30. Show that the following predicate

$$f(x) \overset{\text{def}}{\simeq} \begin{cases} \texttt{True}, & \text{if } \varphi_x(x) \downarrow \\ \uparrow, & \text{if } \varphi_x(x) \uparrow \end{cases}$$

is computable.

Proof. We can define the predicate f in the following way:

$$f(x) \overset{\text{def}}{\simeq} \texttt{true}(\mu z[\mathtt{T}_1(x,x,z) = \texttt{True}]).$$

$\square$

Problem 31. Show that there is *no* computable predicate g such that

$$g(x) = \begin{cases} \texttt{True}, & \text{if } \varphi_x(x)\downarrow \\ \texttt{False}, & \text{if } \varphi_x(x)\uparrow \end{cases}$$

This is an important problem!

Hint. Assume that g is total computable. Then the following function is also computable:

$$h(x) \overset{\text{def}}{\simeq} \begin{cases} \texttt{True}, & \text{if } g(x) = \texttt{False} \\ \uparrow, & \text{if } g(x) = \texttt{True}. \end{cases}$$

There is an index e such that $\varphi_e^{(1)} = h$. Now we have the following chains of implications:

$$\begin{aligned} \varphi_e(e)\downarrow &\implies g(e) = \texttt{True} \implies h(e)\uparrow \implies \varphi_e(e)\uparrow \\ \varphi_e(e)\uparrow &\implies g(e) = \texttt{False} \implies h(e)\downarrow \implies \varphi_e(e)\downarrow . \end{aligned}$$

In both cases, we reach a contradiction. $\square$

Actually, in the proof of *Problem* 31 we solved the following problem.

Problem 32. Show that the following predicate

$$g(x) \overset{\text{def}}{\simeq} \begin{cases} \texttt{True}, & \text{if } \varphi_x(x)\uparrow \\ \uparrow, & \text{if } \varphi_x(x)\downarrow \end{cases}$$

is *not* computable.

Proof. Assume that g is computable. Then let us fix an index e such that $\varphi_e^{(1)} = g$. Following the chains of implications:

$$\begin{aligned} g(e) \simeq \texttt{True} &\implies \varphi_e(e)\uparrow \implies g(e)\uparrow, \\ g(e)\uparrow &\implies \varphi_e(e)\downarrow \implies g(e)\downarrow, \end{aligned}$$

we reach a contradiction. $\square$

Problem 33. Show that there is *no* computable predicate g such that

$$g(x, y) = \begin{cases} \texttt{True}, & \text{if } \varphi_x(y)\downarrow \\ \texttt{False}, & \text{if } \varphi_x(y)\uparrow . \end{cases}$$

Hint. If $g(x, y)$ is total computable, then the function $\hat{g}(x) = g(x, x)$ is also total computable. But this is a contradiction by *Problem* 31. $\square$

Problem 34. Show that if f is computable unary injective function, then f^{-1} is also computable.

Recall that f is injective if for all $x \neq y \implies f(x) \neq f(y)$.

Hint. Since f is computable, there is an index e such that

$$f(x) \simeq \rho(\mu z[\mathtt{T}_1(e, x, z) = \texttt{True}]).$$

Since f is injective, we can define f^{-1} in the following way:

$$f^{-1}(y) \overset{\text{def}}{\simeq} \rho(\mu z[\mathtt{T}_1(e, \lambda(z), \pi(\rho(z), y)) = \texttt{True}]).$$

$\square$

Problem 35. Show that there is a computable function g such that there is no total computable predicate f for which we have

$$g(x) = \mu y[f(x, y) = \texttt{True}].$$

Hint. Consider the computable function g, where

$$g(x) \overset{\text{def}}{\simeq} \begin{cases} x, & \text{if } \varphi_x(x)\downarrow \\ \uparrow, & \text{if } \varphi_x(x)\uparrow . \end{cases}$$

Assume that such total computable predicate f exists. Then it is easy to see that

- If $\varphi_x(x)\downarrow$, then $f(x, x) = \texttt{True}$ and $f(x, y) = \texttt{False}$ for all $y < x$.
- If $\varphi_x(x)\uparrow$, then $f(x, y) = \texttt{False}$ for all y.

It follows that the function $h(x) \overset{\text{def}}{=} f(x, x)$ is also a total computable predicate. But then we can write the definition of h in the following form:

$$h(x) = \begin{cases} \texttt{True}, & \text{if } \varphi_x(x)\downarrow \\ \texttt{False}, & \text{if } \varphi_x(x)\uparrow . \end{cases}$$

We reach a contradiction with *Problem* 31. $\square$

Problem 36. Show that the partial computable function

$$g(x) \simeq \mu y.[\mathtt{T}_1(x, x, y) = \mathtt{True}]$$

cannot be extended to total computable.

Hint. Clearly, g is partial. Assume that $g \subset f$, where f is total computable. Fix an index e such that

$$\varphi_e^{(1)}(x) \overset{\text{def}}{=} f(x) + 1.$$

Since f is total, $\varphi_e(e) \downarrow$ and hence, by the Normal Form Theorem, there is some number y such that $\mathtt{T}_1(e, e, y) = \mathtt{True}$. By the definition of g, $g(e) \downarrow$ and let $g(e) \simeq y$, for some number y. But then, since $g \subset f$, we have that $f(e) \simeq y$ and

$$y = \varphi_e(e) = f(e) + 1 = y + 1.$$

We reach a contradiction. □

Problem 37. Show that there is a primitive recursive function S such that

a) $\varphi_{S(a,b)}(x) \simeq \varphi_a(x) + \varphi_b(x)$;

b) $\varphi_{S(a,b)}(x) \simeq \varphi_a(\varphi_b(x))$;

Problem 38. Let $W_a = \mathtt{dom}(\varphi_a^{(1)})$ and $E_a = \mathtt{rng}(\varphi_a^{(1)})$. Prove that there exist primitive recursive functions such that:

a) $W_{\alpha(a,b)} = W_a \cup \{0, 1, \dots, b\}$;

b) $W_{\gamma(a,b)} = E_a \cap E_b$;

c) $W_{\delta'(a,b)} = W_a \cup E_b$;

d) $W_{\delta''(a,b)} = W_a \cap E_b$;

e) $E_{\rho(a)} = W_a$;

f) $E_{\rho'(a)} = W_a \setminus \{a\}$;

g) $W_{\kappa(a)} = E_a$ and $E_{\kappa(a)} \subseteq W_a$;

h) $E_{\psi(a,b)} = \{x \mid x \in E_a \;\&\; x \geq b\}$;

i) $W_{\theta(a,b)} = \{x \mid \varphi_a(x)\downarrow \;\&\; \varphi_a(x) \in W_b\}$;

j) $W_{\xi(x)} = \{\xi(x) + x\}$;

k) $E_{\xi'(x)} = \{\xi'(x) + x\}$;

Hint.

a) Let us consider the following computable function:

$$f(a, b, x) \simeq \begin{cases} 42, & \text{if } x \leq b \\ \varphi_a(x), & \text{if } x > b. \end{cases}$$

By the Parameters Theorem, there is a primitive recursive α such that for every a, b, and x, $f(a, b, x) \simeq \varphi_{\alpha(a,b)}(x)$. Thus, $W_{\alpha(a,b)} = W_a \cup \{0, 1, \dots, b\}$.

b) Consider the following partial computable function:

$$f(a, b, x) \simeq \begin{cases} 42, & \text{if } (\exists z_1)(\exists z_2)[\varphi_a(z_1) \simeq x \;\&\; \varphi_b(z_2) \simeq x] \\ \uparrow, & \text{otherwise.} \end{cases}$$

j) Consider the following partial computable function:

$$f(a,x,y) \simeq \begin{cases} 42, & \text{if } y \simeq \varphi_a(x) + x \\ \uparrow, & \text{otherwise.} \end{cases}$$

There exists a primitive recursive function S such that for every a and x,

$$W_{S(a,x)} = \{\varphi_a(x) + x\}.$$

By the Recursion Theorem, there exists an index e such that $S(e,x) = \varphi_e(x)$ for every x. Let $\xi = \varphi_e$. Then

$$W_{\xi(x)} = W_{S(e,x)} = \{\varphi_e(x) + x\} = \{\xi(x) + x\}.$$

k) Consider the following partial computable function:

$$f(a,x,y) \simeq \begin{cases} \varphi_a(x) + x, & \text{if } x = y \\ \uparrow, & \text{otherwise.} \end{cases}$$

□

Problem 39. There are infinitely many indices e such that:

(1) $\varphi_e^{(1)} = \varphi_{e+1}^{(1)}$;

(2) $\varphi_e^{(1)} = \varphi_e^{(1)} \circ \varphi_e^{(1)}$;

(3) $\varphi_e^{(1)} = \varphi_{e+1}^{(1)} \circ \varphi_{e+2}^{(1)}$.

Problem 40. Show that there exists a **primitive recursive** function g such that

✍ homework!

$$(\forall a)(\forall e)[\varphi_e^{-1}(W_a) = W_{g(e,a)}].$$

Chapter 4

Decidable and semidecidable sets

4.1 Decidable sets

We say that $A \subseteq \mathbb{N}$ is a **decidable** set if its characteristic function χ_A is computable, where

$$\chi_A(x) \overset{\text{def}}{=} \begin{cases} \texttt{True}, & \text{if } x \in A \\ \texttt{False}, & \text{if } x \notin A. \end{cases}$$

Also called **computable** sets. Of course, every set has a characteristic function. The problem is that it is usually not computable.

Proposition 4.1. If the sets A and B are decidable, then so are the sets

$$A \cap B,\ A \cup B,\ A \setminus B.$$

In other words, the decidable sets are closed under the operations intersection, union, and complement.

Proof. Easy. □

Proposition 4.2. Let A be a decidable set. Then the sets B, C are also decidable, where:

- $B \overset{\text{def}}{=} \{\langle x, y\rangle \mid (\exists z < y)[\langle x, z\rangle \in A]\}$,

- $C \overset{\text{def}}{=} \{\langle x, y\rangle \mid (\forall z < y)[\langle x, z\rangle \in A]\}$.

Proof.

□

It is useful to study an important example of a undecidable set.

Proposition 4.3. The set $K \overset{\text{def}}{=} \{e \in \mathbb{N} \mid \varphi_e(e)\downarrow\}$ is not decidable.

Proof. Assume that the set K is decidable. It follows that its characteristic function

K is called the Kleene set or the diagonal halting set. This set plays an important role in this course.

$$\chi_K(x) = \begin{cases} \texttt{True}, & \text{if } \varphi_x(x)\downarrow \\ \texttt{False}, & \text{if } \varphi_x(x)\uparrow \end{cases}$$

is computable. Now we apply *Problem* 31 to reach a contradiction. □

By *Proposition* 4.1, the complement of the Kleene set, denoted $\overline{K}$ is not decidable. It is a cannonical example, so we will state it explicitly.

Example 5. The complement of the Kleene set, denoted $\overline{K}$, where

$$\overline{K} \overset{\text{def}}{=} \{e \in \mathbb{N} \mid \varphi_e(e)\uparrow\},$$

is not decidable.

Problem 41. Prove that a total function f is computable iff $\texttt{graph}(f)$ is a decidable set.

[4, p. 127]

Proof.

□

Problem 42. Show that there exists a decidable set A such that

- $D \overset{\text{def}}{=} \{x \mid (\exists z)[\langle x, z\rangle \in A]\}$ is **not** decidable, or
- $E \overset{\text{def}}{=} \{x \mid (\forall z)[\langle x, z\rangle \in A]\}$ is **not** decidable.

Proof. Consider the following decidable set:

Recall that Normal Form Theorem says that $\texttt{T}_1$ is the Kleene predicate and that $\texttt{T}_1(a, x, z) = 0$ iff $z = \langle s, y\rangle$ and $\varphi_a(x)\downarrow = y$ for s number of steps.

$$A \overset{\text{def}}{=} \{\langle x, z\rangle \mid \texttt{T}_1(x, x, z) = \texttt{True}\}.$$

Clearly, $\overline{A}$ is also decidable. Then we can show the following:

- $K = \{x \mid (\exists z)[\langle x, z\rangle \in A]\}$;

- $\overline{K} = \{x \mid (\forall z)[\langle x, z\rangle \in \overline{A}]\}$.

□

Problem 43. Let f be a total computable function such that $f(x) \geq x$. Prove that $\mathtt{rng}(f)$ is a decidable set. [4, p. 128]

Proof. Since $(\forall x \in \mathbb{N})[f(x) \geq x]$, to check whether $y \in \mathtt{rng}(f)$, we need to compute only the values $f(x)$ for $x \leq y$. Thus,

$$\begin{aligned} y \in A &\Leftrightarrow (\exists x \leq y)[f(x) = y] \\ &\Leftrightarrow \sum_{x \leq y} \overline{\mathtt{sign}}(|f(x) - y|) = 1. \end{aligned}$$

□

Problem 44. Prove that an *infinite* set A is decidable if and only if $A = \mathtt{rng}(f)$, where f is a computable *total strictly increasing* function. [4, p. 129].

Hint. $(\rightarrow)$ This direction is easy. Consider the function

$$\left| \begin{array}{lcl} f(0) & \overset{\text{def}}{=} & \mu z[\chi_A(z) = \mathtt{True}] \\ f(n+1) & \overset{\text{def}}{=} & \mu z[f(n) < z \;\&\; \chi_A(z) = \mathtt{True}]. \end{array} \right.$$

$(\leftarrow)$ Let $A = \mathtt{rng}(f)$. Since f is strictly increasing, $(\forall z \in \mathbb{N})[f(z) \geq z]$. Then apply *Problem* 43. □

Problems

Problem 45. Let f be total computable increasing function (possibly not strictly). Show that $\mathtt{rng}(f)$ is a decidable set.

Hint. If $\mathtt{rng}(f)$ is finite, then it is clear. If $\mathtt{rng}(f)$ is infinite, then for every y, there is x such that $f(x) \geq y$. □

Problem 46. Let f and g be total computable functions and let g be bijective. Moreover, we require $(\forall x \in \mathbb{N})[f(x) \geq g(x)]$, i.e. f majorizes g. Show that $\mathtt{rng}(f)$ is a decidable set. [4, p. 129]

Hint. Since g is bijective and computable, then g^{-1} is total and computable. Given y, we look for z such that

$$\{0, 1, \ldots, y\} \subseteq \mathtt{rng}(g \restriction \{0, 1, \ldots, z\}).$$

It follows that if $f(x) = y$, then $x < z$. We have to be able to find this z effectively from y. We will define a total computable h such that

$$\{0, 1, \ldots, y\} \subseteq \texttt{rng}(g \restriction \{0, 1, \ldots, h(y) + 1\}).$$

Then $f(x) = y \implies x \leq h(y)$. We define h using the following primitive recursive scheme:

$$\left| \begin{array}{lll} h(0) & \stackrel{\text{def}}{=} & g^{-1}(0) \\ h(y+1) & \stackrel{\text{def}}{=} & \max\{h(y), g^{-1}(y+1)\}. \end{array} \right.$$

Clearly, $h(y) = \max\{g^{-1}(0), \ldots, g^{-1}(y)\}$. Since

$$\begin{array}{rcl} y \in \texttt{rng}(f) & \Leftrightarrow & (\exists x)[\langle x, y\rangle \in \texttt{graph}(f)] \\ & \Leftrightarrow & (\exists x)[\langle x, y\rangle \in \texttt{graph}(f) \ \&\ f(x) = y \geq g(x)] \\ & \Leftrightarrow & (\exists x \leq h(y))[\langle x, y\rangle \in \texttt{graph}(f)], \end{array}$$

if follows that $\texttt{rng}(f)$ is decidable since $\texttt{graph}(f)$ is decidable by *Problem* 41.

□

Problem 47. Let f, g be total computable functions and let g be bijective. Moreover, we require $(\forall x \in \mathbb{N})[f(x) \geq x]$. Show that the set [4, p. 129]

$$A \stackrel{\text{def}}{=} \{g(y) \mid y \in \texttt{rng}(f)\}$$

is decidable.

Hint. Show the following equivalences

$$\begin{array}{rcl} x \in A & \Leftrightarrow & (\exists y)[\langle y, x\rangle \in \texttt{graph}(g) \ \&\ (\exists z \leq y)[\langle z, y\rangle \in \texttt{graph}(f)]] \\ x \notin A & \Leftrightarrow & (\exists y)[\langle y, x\rangle \in \texttt{graph}(g) \ \&\ (\forall z \leq y)[\langle z, y\rangle \notin \texttt{graph}(f)]]. \end{array}$$

□

4.2 Semidecidable sets

We say that $A \subseteq \mathbb{N}$ is a **semidecidable** set if its semicharacteristic function $\hat{\chi}_A$ is computable, where

$$\hat{\chi}_A(x) \overset{\text{def}}{\simeq} \begin{cases} \texttt{True}, & \text{if } x \in A \\ \uparrow, & \text{if } x \notin A. \end{cases}$$

Proposition 4.4. The set A is a semidecidable set if and only if $A = \texttt{dom}(\varphi_a^{(1)})$, for some index a.

Proof. If A is semidecidable, then $\hat{\chi}_A$ is computable, i.e. there exists an index a, $\hat{\chi}_A = \varphi_a$. In this case, $\texttt{dom}(\varphi_a) = A$. Conversely, let $A = \texttt{dom}(\varphi_a)$, for some index a. Then the semicharacteristic function of A is $\hat{\chi}_A = \texttt{true} \circ \varphi_a$. □

We enumerate all semidecidable sets in an infinite sequence,

$$W_0, W_1, \ldots, W_e, \ldots,$$

where $W_e \overset{\text{def}}{=} \texttt{dom}(\varphi_e^{(1)})$.

Proposition 4.5. The set A is semidecidable set if and only if there exists a **primitive recursive** predicate α such that

$$x \in A \;\Leftrightarrow\; (\exists y)[\alpha(x, y) = \texttt{True}].$$

Proof. $(\rightarrow)$ Suppose A is semidecidable. Then $A = \texttt{dom}(\varphi_a)$, for some index a. By Normal Form Theorem, there is a primitive recursive function $\texttt{T}_1$ such that

$$\varphi_a(x) \downarrow \;\Leftrightarrow\; (\exists z)[\texttt{T}_1(a, x, z) = \texttt{True}].$$

We let $\alpha(x, y) \overset{\text{def}}{=} \texttt{T}_1(a, x, y)$.

$(\leftarrow)$ Suppose α is primitive recursive such that

$$x \in A \;\Leftrightarrow\; (\exists y)[\alpha(x, y) = \texttt{True}].$$

Let us define $\varphi(x) \simeq (\mu y)[\alpha(x, y) = \texttt{True}]$. It is clear that φ is computable and $A = \texttt{dom}(\varphi)$. □

Proposition 4.6. Let f be a (partial) function. Then f is computable iff $\texttt{graph}(f)$ is semidecidable.

Proof. ($\rightarrow$) Suppose f is computable. Then $f = \varphi_a$, for some a. By Normal Form Theorem, the function $\mathtt{T}_1$ is primitive recursive and

$$f(x) \simeq y \;\Leftrightarrow\; (\exists z)[\mathtt{T}_1(a,x,z) = \mathtt{True} \wedge y = \rho(z)].$$

($\leftarrow$) Suppose $\mathtt{graph}(f)$ is semidecidable. There exists a primitive recursive predicate γ such that

$$\langle x, y\rangle \in \mathtt{graph}(f) \;\Leftrightarrow\; f(x) \simeq y \;\Leftrightarrow\; (\exists z)[\gamma(x,y,z) = \mathtt{True}].$$

Thus,

$$f(x) \simeq \lambda(\mu t[\gamma(x, \lambda(t), \rho(t)) = \mathtt{True}]).$$

□

Proposition 4.7. Let A be a **non-empty** semidecidable set. Then there exists a **primitive recursive** function g such that $A = \mathtt{rng}(g)$.

In other words, there is a total computable function g which *enumerates* the elements of A in some arbitrary order, possibly with repetitions.

Proof. By *Proposition* 4.5, there is a primitive recursive predicate α such that

$$x \in A \;\Leftrightarrow\; (\exists y \in \mathbb{N})[\alpha(x,y) = \mathtt{True}].$$

Fix $a \in A$. Define the primitive recursive function g:

$$g(n) \overset{\text{def}}{=} \begin{cases} a, & \text{if } \alpha(\lambda(n), \rho(n)) = \mathtt{False} \\ \lambda(n), & \text{if } \alpha(\lambda(n), \rho(n)) = \mathtt{True}. \end{cases}$$

It is easy to see that $A = \mathtt{rng}(g)$. □

Theorem 4.1 (Post)**.** The set A is decidable if and only if A and its complement $\overline{A} = \mathbb{N} \setminus A$ are semidecidable sets.

Proof. Let both A and $\overline{A}$ be semidecidable. By *Proposition* 4.5, there exist primitive recursive predicates α and $\overline{\alpha}$ such that

$$\begin{aligned} x \in A \quad &\Leftrightarrow \quad \exists y[\alpha(x,y) = \mathtt{True}] \\ x \in \overline{A} \quad &\Leftrightarrow \quad \exists y[\overline{\alpha}(x,y) = \mathtt{True}]. \end{aligned}$$

The function

$$h(x) \overset{\text{def}}{=} \mu y[\alpha(x,y) = \mathtt{True} \;\vee\; \overline{\alpha}(x,y) = \mathtt{True}]$$

is total and computable. Thus,

$$\chi_A(x) = \texttt{sign}(\alpha(x, h(x))).$$

The other direction is obvious. □

Notice that *Problem* 30 tells us that the semicharacteristic function $\hat{\chi}_K$ is computable and hence K is semidecidable. On the other hand, *Problem* 32 tells us that $\overline{K}$ is not semidecidable.

Example 6. The Kleene set $K \stackrel{\text{def}}{=} \{x \mid \varphi_x(x) \downarrow\}$ is semidecidable, but **not** decidable. Its complement $\overline{K}$ is not semidecidable.

Proposition 4.8. If A and B are semidecidable sets, then $A \cap B$ and $A \cup B$ are also semidecidable, but $A \setminus B$ may not be semidecidable.

In other words, the semidecidable sets are closed under the operations of intersection and union, but not under the operation complement.

Proposition 4.9. Let A be a semidecidable set. Then the sets B, C and D are also semidecidable sets:

- $B \stackrel{\text{def}}{=} \{\langle \overline{x}, y\rangle \mid (\exists z < y)[\langle \overline{x}, z\rangle \in A]\}$,
- $C \stackrel{\text{def}}{=} \{\langle \overline{x}, y\rangle \mid (\forall z < y)[\langle \overline{x}, z\rangle \in A]\}$,
- $D \stackrel{\text{def}}{=} \{\overline{x} \mid (\exists y)[(\overline{x}, y) \in A]\}$.

It is possible that the set E is not semidecidable, where:

$$E \stackrel{\text{def}}{=} \{\langle \overline{x}\rangle \mid (\forall y)[\langle \overline{x}, y\rangle \in A]\}.$$

Hint. By *Proposition* 4.5, let α be a primitive recursive predicate such that

$$(\overline{x}, y) \in A \iff (\exists u)[\alpha(\overline{x}, y, u) = \texttt{True}].$$

Prove that the following hold:

- $\langle \overline{x}, y\rangle \in B \iff (\exists u)[\sum_{z<y} \alpha(\overline{x}, z, u) \geq 1]$.
- $\langle \overline{x}, y\rangle \in C \iff (\exists u)[\prod_{z<y} \alpha(\overline{x}, z, (u)_z) = \texttt{True}]$.
- $\overline{x} \in D \iff (\exists t)[\alpha(\overline{x}, \lambda(t), \rho(t)) = \texttt{True}]$.

□

Proposition 4.10. The set A is semidecidable if and only if there exists a decidable set B such that the following holds:

$$x \in A \;\Leftrightarrow\; (\exists y)[\langle x, y\rangle \in B].$$

Proof. $(\rightarrow)$ Suppose A is semidecidable. By *Proposition* 4.5, there exists a primitive recursive α such that

$$x \in A \quad\Leftrightarrow\quad (\exists y)[\alpha(x, y) = \texttt{True}].$$

Choose the decidable set B such that $\chi_B = \alpha$.

$(\leftarrow)$ Let B be decidable. Then clearly B is also semidecidable and hence A is semidecidable from *Proposition* 4.9. □

Problems

Problem 48. Show that the following structure

$$\mathcal{E} = (\{W_e\}_{e\in\mathbb{N}}, \cup, \cap, \mathbb{N}, \emptyset)$$

is a distributive lattice.

Problem 49. Let A be an *infinite* semidecidable set. Prove that there exists an injective total computable function h such that $A = \texttt{rng}(h)$.

We will use this fact later. [4, p. 135, Problem 21]

Hint. Let g be primitive recursive such that $A = \texttt{rng}(g)$. Prove that the following function h has all needed properties:

We use *Proposition* 4.7.

$$\left|\begin{array}{lcl} h(0) & \overset{\text{def}}{=} & g(0) \\ h(n+1) & \overset{\text{def}}{=} & g(\mu z[\prod_{i\le n} |h(i) - g(z)| \neq 0]). \end{array}\right.$$

□

Problem 50. Prove that every infinite semidecidable set contains an infinite decidable subset.

[4, p. 136, Problem 22]

Hint. Let $A = \texttt{rng}(h)$, where h is total computable. We define a strictly increasing total computable f such that $\texttt{rng}(f) \subset A$. Fix some element $a_0 \in A$. It is easy to see that the function f has the necessary properties, where:

Then we apply Problem 44

$$\left|\begin{array}{lcl} f(0) & \overset{\text{def}}{=} & a_0 \\ f(n+1) & \overset{\text{def}}{=} & h(\mu z[(f(n)+1) \dot{-} h(z) = 0]). \end{array}\right.$$

□

4.3 Decidable index sets

Let $\mathscr{C}$ be a class of unary computable functions. The **index set** for $\mathscr{C}$ is the set

$$I_{\mathscr{C}} \overset{\text{def}}{=} \{e \mid \varphi_e^{(1)} \in \mathscr{C}\}.$$

For example, the set $\texttt{Empty} \overset{\text{def}}{=} \{e \mid \varphi_e^{(1)} = \emptyset^{(1)}\}$ is an index set, where $\mathscr{C} = \{\emptyset^{(1)}\}$. On the other hand, the set $K = \{e \mid \varphi_e^{(1)}(e) \downarrow\}$ is not an index set. Here we will study the problem how to determine whether a given index set is computable or not.

Problem 51. Show that the set $\texttt{Empty}$ is **not** decidable.

Later we will see that this problem is a direct corollary of Rice Theorem.

Proof. Firstly, let us fix an index e_0 such that $e_0 \in \texttt{Empty}$ and let e_1 be an index such that $e_1 \notin \texttt{Empty}$. Assume $\texttt{Empty}$ is decidable and consider the total computable function f, where

$$f(x) \overset{\text{def}}{\simeq} \begin{cases} e_1, & \text{if } x \in \texttt{Empty} \\ e_0, & \text{if } x \notin \texttt{Empty}. \end{cases}$$

By the Fixed Point Theorem, there exists an index a such that

This is a form of diagonalization.

$$\varphi_a^{(1)} = \varphi_{f(a)}^{(1)}.$$

Then we have the following chains of implications:

$$a \in \texttt{Empty} \implies \varphi_a^{(1)} = \emptyset^{(1)} \implies \varphi_{f(a)}^{(1)} = \emptyset^{(1)} \implies \varphi_{e_1}^{(1)} = \emptyset^{(1)} \implies e_1 \in \texttt{Empty}.$$

$$a \notin \texttt{Empty} \implies \varphi_a^{(1)} \neq \emptyset^{(1)} \implies \varphi_{f(a)}^{(1)} \neq \emptyset^{(1)} \implies \varphi_{e_0}^{(1)} \neq \emptyset^{(1)} \implies e_0 \notin \texttt{Empty}.$$

In both cases, we reach a contradiction. □

Proof. Assume that $\texttt{Empty}$ is decidable and consider the following computable function

This proof gives us more information. We can even assume that $\texttt{Empty}$ is semidecidable. We will reach a contradiction in the same way. It follows that $\texttt{Empty}$ is not semidecidable.

$$f(x, y) \overset{\text{def}}{\simeq} \begin{cases} 5, & \text{if } x \in \texttt{Empty} \\ \uparrow, & \text{if } x \notin \texttt{Empty}. \end{cases}$$

By the Recursion Theorem, there exists an index e such that

$$\varphi_e^{(1)}(y) \simeq f(e, y).$$

Then we have the following chains of implications:

$$\begin{aligned} e \in \texttt{Empty} &\implies \varphi_e^{(1)} \text{ is total} &&\implies e \notin \texttt{Empty} \\ e \notin \texttt{Empty} &\implies \varphi_e^{(1)} = \emptyset^{(1)} &&\implies e \in \texttt{Empty}. \end{aligned}$$

We reach a contradiction. □

Proof. We shall effectively reduce K to $\overline{\texttt{Empty}}$, the complement of `Empty`. Define the function

This proof presents the most general idea: effectively reducing one set to another.

$$f(x,y) \simeq \begin{cases} 42, & \text{if } x \in K \\ \uparrow, & \text{if } x \notin K. \end{cases}$$

Clearly, f is computable. By the Parameters Theorem, there exists a primitive recursive h such that $f(x,y) \simeq \varphi_{h(x)}^{(1)}(y)$. Then

$$x \in K \implies \varphi_{h(x)} \neq \emptyset^{(1)} \implies h(x) \in \overline{\texttt{Empty}}.$$

$$x \notin K \implies \varphi_{h(x)} = \emptyset^{(1)} \implies h(x) \notin \overline{\texttt{Empty}}.$$

It follows that

$$K = \{x \mid h(x) \in \overline{\texttt{Empty}}\}.$$

If we assumed that `Empty` is decidable, then $\overline{\texttt{Empty}}$ would be decidable and hence K would be decidable. A contradiction. □

Problem 52. Show that the set

$$\texttt{Quine} \overset{\text{def}}{=} \{a \mid (\forall x)[\varphi_a^{(1)}(x) \simeq a]\}$$

is not decidable.

✍ Explain why `Quine` is not an index set! Later, we will show that `Quine` is not even semidecidable.

Proof. Assume that `Quine` is decidable. Consider the computable function

$$f(x,y) \overset{\text{def}}{\simeq} \begin{cases} \uparrow, & \text{if } x \in \texttt{Quine} \\ x, & \text{if } x \notin \texttt{Quine}. \end{cases}$$

By the Recursion Theorem, there exists an index e such that

$$\varphi_e^{(1)}(y) \simeq f(e,y).$$

$$\begin{aligned} e \in \texttt{Quine} &\implies (\forall y)[f(e,y)\uparrow] \implies \varphi_e = \emptyset^{(1)} \implies e \notin \texttt{Quine} \\ e \notin \texttt{Quine} &\implies (\forall y)[f(e,y) \simeq e] \implies (\forall y)[\varphi_e(y) \simeq e] \implies e \in \texttt{Quine}. \end{aligned}$$

We reach a contradiction. □

Problem 53. Show that the set

✍ Explain why this is not an index set!

$$A \overset{\text{def}}{=} \{\ a \mid W_a = \{a\}\ \}$$

is not decidable.

Hint. Assume that A is decidable and consider the computable function

$$f(x, y) \simeq \begin{cases} 5, & \text{if } x \notin A \ \&\ x = y \\ \uparrow, & \text{otherwise.} \end{cases}$$

Proceed as above. $\square$

Problem 54. Show that the set

$$B \overset{\text{def}}{=} \{\ a \mid W_a = \mathbb{N} \setminus \{a\}\ \}$$

is not decidable.

Theorem 4.2 (Rice 1953)**.** Let $\mathscr{C}$ be a class of unary computable functions. The index set $I_{\mathscr{C}}$ is **decidable** if and only if $\mathscr{C} = \emptyset$ or when $\mathscr{C}$ is the class of all computable functions.

Proof. It is clear that if $\mathscr{C}$ is trivial, then $I_{\mathscr{C}}$ is decidable. We will prove the direction ($\Longrightarrow$) by using contraposition, i.e. we will prove that if $\mathscr{C}$ is *not* trivial, then $I_{\mathscr{C}}$ is *not* decidable. Assume that there are computable functions ψ_0 and ψ_1 such that $\psi_0 \notin \mathscr{C}$ and $\psi_1 \in \mathscr{C}$. Consider the computable function

$$f(x, y) \overset{\text{def}}{\simeq} \begin{cases} \psi_0(y), & \text{if } x \in I_{\mathscr{C}} \\ \psi_1(y), & \text{if } x \notin I_{\mathscr{C}}. \end{cases}$$

By the Recursion Theorem, there exists an index e such that

$$\varphi_e^{(1)}(y) \simeq f(e, y).$$

Now we obtain the following chains of implications:

$$\begin{aligned} e \in I_{\mathscr{C}} &\implies \varphi_e = \psi_0 \implies e \notin I_{\mathscr{C}} \\ e \notin I_{\mathscr{C}} &\implies \varphi_e = \psi_1 \implies e \in I_{\mathscr{C}}. \end{aligned}$$

We reach a contradiction. $\square$

Proof. It is clear that if $\mathscr{C} = \emptyset$ or when $\mathscr{C}$ contains all computable functions, then $I_{\mathscr{C}}$ is computable. Now assume $I_{\mathscr{C}}$ is a computable set and fix $\varphi_a \in \mathscr{C}$ and $\varphi_b \notin \mathscr{C}$. Define the total computable function f in the following way:

$$f(x) = \begin{cases} b, \text{ if } x \in I_{\mathscr{C}} \\ a, \text{ if } x \notin I_{\mathscr{C}} \end{cases}$$

We have that

$$f(x) \in I_{\mathscr{C}} \;\Leftrightarrow\; f(x) = a \;\Leftrightarrow\; x \notin I_{\mathscr{C}}.$$

By the Fixed Point Theorem, there exists an index e, such that $\varphi_e^{(1)} \simeq \varphi_{f(e)}^{(1)}$. Thus,

$$\varphi_e^{(1)} \in \mathscr{C} \;\Leftrightarrow\; \varphi_e^{(1)} \notin \mathscr{C},$$

which is a contradiction. □

Proof. It is clear that if $\mathscr{C} = \emptyset$ or when $\mathscr{C}$ contains all computable functions, then $I_{\mathscr{C}}$ is decidable. We shall define a total computable function h such that

$$x \in K \;\Leftrightarrow\; h(x) \in I_{\mathscr{C}}.$$

Suppose that $\emptyset^{(1)} \notin \mathscr{C}$.

Since K is semidecidable, then the following function is partial computable

$$f(x, y) \overset{\text{def}}{\simeq} \begin{cases} \psi(y), & \text{if } x \in K \\ \uparrow, & \text{otherwise.} \end{cases}$$

By the Parameters Theorem, there exists a primitive recursive, and consequently total computable, function h such that

$$f(x, y) \simeq \varphi_{h(x)}^{(1)}(y),$$

for all natural numbers x and y. Then

$$\begin{aligned} x \in K &\implies \varphi_{h(x)}^{(1)} = \psi \implies h(x) \in I_{\mathscr{C}} \\ x \notin K &\implies \varphi_{h(x)}^{(1)} = \emptyset^{(1)} \implies h(x) \notin I_{\mathscr{C}}. \end{aligned}$$

If $\emptyset^{(1)} \in \mathscr{C}$, then we can consider the complement $\overline{\mathscr{C}}$ of $\mathscr{C}$ and show that there is a total computable function h such that

$$x \in K \;\Leftrightarrow\; h(x) \in I_{\overline{\mathscr{C}}}.$$

Then $I_{\overline{\mathscr{C}}}$ is not decidable and since $I_{\mathscr{C}} = \mathbb{N} \setminus I_{\overline{\mathscr{C}}}$, if follows that $I_{\mathscr{C}}$ is not a deciable set. □

Example 7. As a direct corollary of the Rice Theorem, the following index sets are not decidable:

a) $\texttt{Empty} = \{a \mid \varphi_a^{(1)} = \emptyset^{(1)}\}$;

b) $\texttt{Fin} = \{a \mid \mathit{Dom}(\varphi_a^{(1)}) \text{ is finite}\}$;

c) $\texttt{Inf} = \{a \mid \mathit{Dom}(\varphi_a^{(1)}) \text{ is infinite}\}$;

d) $\texttt{Tot} = \{a \mid \varphi_a^{(1)} \text{ is total}\}$;

e) $\texttt{Const} = \{a \mid \varphi_a^{(1)} \text{ is a constant function}\}$;

f) $\texttt{Eq}_a = \{x \mid \varphi_x^{(1)} = \varphi_a^{(1)}\}$;

g) $\texttt{Eq} = \{\langle x, y\rangle \mid \varphi_x^{(1)} = \varphi_y^{(1)}\}$.

4.4 Semidecidable index sets

Problem 55. Show that the index set for the class $\{\emptyset^{(1)}\}$, denoted

$$\texttt{Empty} \overset{\text{def}}{=} \{a \mid \varphi_a^{(1)} = \emptyset^{(1)}\},$$

is **not** semidecidable.

Proof. Here we essentially repeat the third proof of the fact that Empty is not decidable. There we proved that

Recall *Example* 6 which says that $\overline{K}$ is not semidecidable.

$$K = \{x \mid h(x) \in \overline{\texttt{Empty}}\}.$$

But this is equivalent to the following:

$$\overline{K} = \{x \mid h(x) \in \texttt{Empty}\}.$$

If we assumed that Empty is semidecidable, then $\overline{K}$ would be semidecidable, which is evidently not true. □

We already know that the set Tot - the index set of total computable functions is not decidable. Now we will prove that Tot is not even semidecidable. The proof is important because we will use the same proof idea again in a while.

Problem 56. Show that the index set for the class of total unary computable functions, denoted

$$\texttt{Tot} \overset{\text{def}}{=} \{a \mid \varphi_a^{(1)} \text{ is total}\},$$

is **not** semidecidable.

Proof. We will show that there is a total computable function h such that

$$x \in \overline{K} \iff h(x) \in \texttt{Tot}.$$

Since the set K is semidecidable, consider the primitive recursive predicate κ such that we have

$$x \in K \iff (\exists y)[\kappa(x, y) = \texttt{True}].$$

Define the computable function g in the following way:

$$g(x,y) \overset{\text{def}}{\simeq} \begin{cases} 42, & \text{if } (\forall z \le y)[\kappa(x,z) = \texttt{False}], \\ \uparrow, & \text{if } (\exists z \le y)[\kappa(x,z) = \texttt{True}]. \end{cases}$$

By the Parameters Theorem, we can find a primitive recursive function h such that for every x and y, $g(x,y) \simeq \varphi^{(1)}_{h(x)}(y)$. Then

$$x \in K \implies (\exists y)[\kappa(x,y_0) = \texttt{True}] \implies \texttt{dom}(\varphi^{(1)}_{h(x)}) \text{ is finite} \implies h(x) \notin \texttt{Tot}$$
$$x \notin K \implies (\forall y)[\kappa(x,y_0) = \texttt{False}] \implies \varphi^{(1)}_{h(x)} \text{ is total} \implies h(x) \in \texttt{Tot}.$$

□

Theorem 4.3 (Rice-Shapiro)**.** Let $\mathscr{C}$ be a class of unary computable functions, for which $I_{\mathscr{C}}$ is a **semidecidable** set. Then for every computable function f, we have

$$f \in \mathscr{C} \Leftrightarrow (\exists \theta \subseteq f)[\ \theta \in \mathscr{C}\ \&\ \theta \text{ is finite}\ ].$$

Proof. ($\Rightarrow$). Let $f \in \mathscr{C}$, but assume $(\forall \theta \subseteq f)[\ \theta \notin \mathscr{C}\ ]$. Since the set K is semidecidable, consider the primitive recursive predicate κ such that we have

We always use θ to denote finite functions.

$$x \in K \Leftrightarrow (\exists y)[\kappa(x,y) = \texttt{True}].$$

Define the computable function g in the following way:

$$g(x,y) \overset{\text{def}}{\simeq} \begin{cases} f(y), & \text{if } (\forall z \le y)[\kappa(x,z) = \texttt{False}], \\ \uparrow, & \text{if } (\exists z \le y)[\kappa(x,z) = \texttt{True}]. \end{cases}$$

Let a be an index for g. By the Parameters Theorem, we can find a primitive recursive function h such that for every x and y,

$$g(x,y) \simeq \varphi^{(1)}_{S^1_1(a,x)}(y) \simeq \varphi^{(1)}_{h(x)}(y).$$

Our goal is to show that $(\forall x)[\ x \in \overline{K} \Leftrightarrow h(x) \in I_{\mathscr{C}}\ ]$. It will follow that $\overline{K}$ is semidecidable, which is evidently not true.

By the definition of g, for every x, $\varphi^{(1)}_{h(x)} \subseteq f$. Now we have two cases to consider.

- If $x \in K$, then $\kappa(x, y_0) = \texttt{True}$, for some least y_0. Then $(\forall y \geq y_0)[\, g(x,y)\uparrow\,]$. Thus, $\varphi^{(1)}_{h(x)}$ is a finite function $\subseteq f$. Since we assumed that $(\forall \theta \subseteq f)[\, \theta \notin \mathscr{C}\,]$, we have $h(x) \notin I_{\mathscr{C}}$.

- If $x \notin K$, then $(\forall y)[\, \kappa(x,y) = \texttt{False}\,]$ and $\varphi^{(1)}_{h(x)} = f$. Thus, $h(x) \in I_{\mathscr{C}}$.

Thus, we conclude

$$x \in \overline{K} \;\Leftrightarrow\; x \notin K \;\Leftrightarrow\; h(x) \in I_{\mathscr{C}}.$$

Since $I_{\mathscr{C}}$ is semidecidable and h is computable, it follows that $\overline{K}$ is semidecidable We reach a contradiction. Thus, our assumption is incorrect.

($\Leftarrow$). Let $f \notin \mathscr{C}$ be a computable function, but assume that there exists $\theta \subseteq f$ such that $\theta \in \mathscr{C}$. This time we define the function g in the following way:

$$g(x,y) \overset{\text{def}}{\simeq} \begin{cases} f(y), & \text{if } \theta(y)\downarrow \,\vee\, x \in K \\ \uparrow, & \text{otherwise} \end{cases}$$

Since f and θ are computable, and K is semidecidable, the function g is also computable. Again by the Parameters Theorem, we take a primitive recursive function h such that for every x and y, $g(x,y) \simeq \varphi^{(1)}_{h(x)}(y)$. We have the following for every x:

$$\begin{array}{lclcl} x \in K & \Longrightarrow & \varphi^{(1)}_{h(x)} = f & \Longrightarrow & h(x) \notin I_{\mathscr{C}} \\ x \notin K & \Longrightarrow & \varphi^{(1)}_{h(x)} = \theta & \Longrightarrow & h(x) \in I_{\mathscr{C}}. \end{array}$$

In the end, $\overline{K} = \{x \mid h(x) \in I_{\mathscr{C}}\}$. Since $I_{\mathscr{C}}$ is semidecidable and h is computable, $\overline{K}$ is semidecidable, which is a contradiction. □

Corollary 4.1 (Rice's theorem)**.** Let $\mathscr{C}$ be a class of computable unary functions. The index set $I_{\mathscr{C}}$ is decidable iff the class $\mathscr{C}$ is either empty or contains all computable unary functions.

This corollary shows that the Rice-Shapiro Theorem is more powerful than the Rice Theorem.

Proof. ($\rightarrow$) Let $I_{\mathscr{C}}$ be decidable, but assume that $\mathscr{C}$ is a nontrivial class. We have that both $I_{\mathscr{C}}$ and $I_{\bar{\mathscr{C}}}$ are semidecidable sets. We consider two cases:

(i) $\emptyset^{(1)} \in \mathscr{C}$. By the previous corollary, every computable function is in $\mathscr{C}$. Thus, $\mathscr{C}$ is a trivial class.

(ii) $\emptyset^{(1)} \notin \mathscr{C}$. Then $\emptyset^{(1)} \in \bar{\mathscr{C}}$ and this time we have that $\bar{\mathscr{C}}$ is a trivial class, but then so is $\mathscr{C}$.

($\leftarrow$) This direction is immediate. □

Corollary 4.2. Let $\mathscr{C}$ be a class of computable unary functions and $I_{\mathscr{C}}$ is semidecidable. If $f \in \mathscr{C}$, then every computable g which extends f belongs to $\mathscr{C}$.

Example 8. As a direct corollary of the Rice-Shapiro Theorem, the following index sets are **not** semidecidable:

a) $\mathtt{Empty} = \{a \mid \varphi_a^{(1)} = \emptyset^{(1)}\}$;

b) $\mathtt{Fin} = \{a \mid \mathit{Dom}(\varphi_a^{(1)}) \text{ is finite}\}$;

c) $\mathtt{Inf} = \{a \mid \mathit{Dom}(\varphi_a^{(1)}) \text{ is infinite}\}$;

d) $\mathtt{Tot} = \{a \mid \varphi_a^{(1)} \text{ is total}\}$;

e) $\mathtt{Const} = \{a \mid \varphi_a^{(1)} \text{ is a constant function}\}$;

f) $\mathtt{Eq}_a = \{x \mid \varphi_x^{(1)} = \varphi_a^{(1)}\}$;

g) $\mathtt{Eq} = \{\langle x, y\rangle \mid \varphi_x^{(1)} = \varphi_y^{(1)}\}$.

4.4.1 Problems

Problem 57. Let ψ is an arbitrary computable unary function. Show that the index set $I_{\{\psi\}}$ is **not** semidecidable.

Problem 58. Show that the index set

$$\mathtt{Prim} = \{e \mid \varphi_e \text{ is a primitive recursive funciton}\}$$

is not semidecidable.

Problem 59. Show that the set

$$\mathtt{Quine} \overset{\text{def}}{=} \{a \mid (\forall x)[\varphi_a^{(1)}(x) \simeq a]\}$$

is not semidecdiable.

Now it is clear to us that this is not an index set, so we cannot apply the Rice-Shapiro Theorem.

Proof. Assume that the set $\mathtt{Quine}$ is semidecidable. We know that there exists a primitive recursive predicate κ such that

Compare with *Problem* 52.

$$x \in K \;\Leftrightarrow\; (\exists u)[\kappa(x, u) = \mathtt{True}].$$

Consider the computable function

$$f(x,y,z) \overset{\text{def}}{\simeq} \begin{cases} S^1_1(x,y), & \text{if } (\forall u < z)[\kappa(y,u) = \texttt{False}] \\ \uparrow, & \text{otherwise} \end{cases}$$

By the Recursion Theorem, there exists an index e such that for all y and z,

$$\varphi^{(2)}_e(y,z) \simeq f(e,y,z).$$

Now we apply the Parameters Theorem and obtain $h(y) = S^1_1(e,y)$ such that for all y and z,

$$f(e,y,z) \simeq \varphi^{(1)}_{h(y)}(z)$$

Then we can conclude that

$$\begin{aligned} x \in \overline{K} &\implies (\forall z)[\varphi^{(1)}_{h(y)}(z) \simeq h(y)] \implies h(y) \in \texttt{Quine} \\ x \notin \overline{K} &\implies \varphi^{(1)}_{h(y)} \text{ is finite} \implies h(y) \notin \texttt{Quine}. \end{aligned}$$

It follows that we have the equivalence:

$$x \in \overline{K} \iff h(x) \in \texttt{Quine},$$

which means that $\overline{K}$ is semidecidable. We reach a contradiction. □

Problem 60. Show that the set

The set A is not an index set! ✍ Explain why.

$$A \overset{\text{def}}{=} \{\, a \mid W_a = \{a\} \,\}$$

is not semidecidable.

Hint. Use the computable function

Compare with *Problem* 53.

$$f(x,y,z) \overset{\text{def}}{\simeq} \begin{cases} 5, & \text{if } x \in K \ \lor \ z = S^1_1(x,y) \\ \uparrow, & \text{otherwise} \end{cases}$$

to show that there exists a total computable function h such that

$$x \in \overline{K} \iff h(x) \in A.$$

□

Hint. Assume that A is semidecidable. Use the computable function

$$f(x,y) \simeq \begin{cases} 5, & \text{if } x \in A \ \lor \ x = y \\ \uparrow, & \text{otherwise} \end{cases}$$

to reach a contradiction. □

Problem 61. Show that the following sets are not semidecidable.

These are not index sets, so we cannot apply the Rice-Shapiro Theorem.

a) $\{a \in \mathbb{N} \mid W_a \neq \{a\}\}$;

b) $\{a \in \mathbb{N} \mid W_a = \mathbb{N} \setminus \{a\}\}$;

c) $\{a \in \mathbb{N} \mid |W_a| = a\}$;

d) $\{a \in \mathbb{N} \mid |W_a| \neq a\}$;

e) $\{a \in \mathbb{N} \mid W_a = \{0, 1, \dots, a\}\}$;

4.4.2 Theorem of McNaughton-Myhill-Rice-Shapiro

Notice that the condition $f \in \mathscr{C} \iff (\exists \theta \subseteq f)[\theta \in \mathscr{C}]$ is a necessary, but not sufficient condition for the index set $I_{\mathscr{C}}$ to be semidecidable. There exist classes $\mathscr{C}$ such that $f \in \mathscr{C} \iff (\exists \theta \subseteq f)[\theta \in \mathscr{C}]$, but $I_{\mathscr{C}}$ is **not** semidecidable.

Proposition 4.11. There exists a class $\mathscr{C}$ of computable functions such that $I_{\mathscr{C}}$ is not semidecidable, but

$$f \in \mathscr{C} \iff (\exists \theta \subseteq f)[\theta \text{ is finite and } \theta \in \mathscr{C}].$$

Hint. Let us consider the complement of the Kleene set

$$\overline{K} = \{k_0 < k_1 < \cdots < k_n < \cdots\}.$$

Define θ_n as the finite function such that $\mathtt{graph}(\theta_n) = \{\langle 0, k_n \rangle\}$. Define the class of computable functions

$$\mathscr{C} = \{\varphi \mid (\exists n)[\theta_n \subseteq \varphi]\}.$$

Then

$$x \in \overline{K} \iff (\exists e)[e \in I_{\mathscr{C}} \mathbin{\&} \varphi_e(0) \simeq x].$$

We conclude that $I_{\mathscr{C}}$ is not semidecidable. □

For a finite function θ, define the **code** of θ as

$$\ulcorner \theta \urcorner \overset{\text{def}}{=} \prod_{i \in \mathtt{dom}(\theta)} p_i^{\theta(i)+1} \dot{-} 1.$$

Proposition 4.12. The set $A = \{\langle x, e \rangle \mid x = \ulcorner \theta \urcorner \mathbin{\&} \theta \subseteq \varphi_e\}$ is semidecidable.

Hint. Here we use the Kleene predicate $\mathtt{T}_1$ from the Normal form theorem. If $x = \ulcorner\theta\urcorner$, then $\theta \subseteq \varphi_e$ iff

$$(\exists s)(\forall i < x)[(x+1)_i = 0 \vee \mathtt{T}_1(e, i, \pi(s, (x+1)_i \dot{-} 1)) = \mathtt{True}].$$

□

Theorem 4.4 (McNaughton-Myhill-Rice-Shapiro)**.** Let $\mathscr{C}$ be a class of computable unary functions. Then $I_{\mathscr{C}}$ is semidecidable **iff** there exists a semidecidable set E of codes of finite functions such that for every function f,

$$f \in \mathscr{C} \;\Leftrightarrow\; (\exists\theta \subseteq f)[\,\ulcorner\theta\urcorner \in E\,].$$

Proof. $(\rightarrow)$ Suppose $I_{\mathscr{C}}$ is semidecidable. We have to show that there is an effective way to find, given the code $\ulcorner\theta\urcorner$, a computable index e such that $\theta = \varphi_e$. More precisely, we will find a primitive recursive function σ such that

[4, p. 160]. Everywhere we use the letter θ to denote finite functions

$$\theta = \varphi^{(1)}_{\sigma(\ulcorner\theta\urcorner)}.$$

Consider the computable function g, where

$$g(x,y) \overset{\text{def}}{\simeq} \begin{cases} z, & \text{if } (x+1)_y = z+1 \\ \uparrow, & \text{if } (x+1)_y = 0. \end{cases}$$

Clearly, we have the equivalence

$$g(x,y) \simeq z \;\Leftrightarrow\; (\exists\theta)[\ulcorner\theta\urcorner = x \;\&\; \theta(y) = z].$$

By the Parameters Theorem, there exists a primitive recursive function σ such that for every x and y,

We have to show that we can effectively go from the number $\ulcorner\theta\urcorner$ to the number a such that $\varphi_a = \theta$

$$g(x,y) \simeq \varphi^{(1)}_{\sigma(x)}(y).$$

It follows that σ *translates* a code of a finite function into the URM program index for the same finite function. In other words,

$$\varphi^{(1)}_{\sigma(\ulcorner\theta\urcorner)} = \theta.$$

Now we consider the set

The empty function $\emptyset^{(1)}$ is obviously a finite function

$$E \overset{\text{def}}{=} \{x \mid \sigma(x) \in I_{\mathscr{C}}\}.$$

Since $I_{\mathscr{C}}$ is semidecidable and σ is computable, the set E is a semidecidable set. To finish the proof of this direction, we have to consider the following two cases.

- Let $f \in \mathscr{C}$. Then by the Rice-Shapiro Theorem, there is some finite $\theta \subseteq f$ such that $\theta \in \mathscr{C}$. Since $\varphi^{(1)}_{\sigma(\ulcorner\theta\urcorner)} = \theta$, it follows that $\sigma(\ulcorner\theta\urcorner) \in I_{\mathscr{C}}$ and hence $\ulcorner\theta\urcorner \in E$.

- Now let $\theta \subseteq f$ and $\ulcorner\theta\urcorner \in E$. Then $\sigma(\ulcorner\theta\urcorner) \in I_{\mathscr{C}}$ and hence $\theta \in \mathscr{C}$. Again by the Rice-Shapiro Theorem, the function $f \in \mathscr{C}$.

($\leftarrow$) Let E be a semidecidable set of codes of finite functions such that

$$f \in \mathscr{C} \;\Leftrightarrow\; (\exists \theta \subseteq f)[\ulcorner\theta\urcorner \in E].$$

We can represent the index set $I_{\mathscr{C}}$ in the following way:

$$I_{\mathscr{C}} = \{a \mid (\exists\theta)[\ulcorner\theta\urcorner \in E \;\&\; \theta \subseteq \varphi^{(1)}_a]\}.$$

It is easy to see that $I_{\mathscr{C}}$ is semidecidable. $\square$

Why is $I_{\mathscr{C}}$ semidecidable?

4.5 Problems

Consider an arbitrary decidable set A. There exists an index a such that $A = W_a$ and an index b such that $\overline{A} = W_b$. It is a natural question to ask whether we can *effectively* obtain the index b from the index a, or vice versa. The next problem tells us that we generally cannot do this.

Problem 62. There is no computable function f such that if W_a is decidable, then $f(a)\downarrow$ and $W_{f(a)} = \overline{W}_a$.

Hint. Let h be total computable such that

$$W_{h(x)} = \begin{cases} \mathbb{N}, & \text{if } x \in K \\ \emptyset, & \text{if } x \notin K. \end{cases}$$

Then the complement of the Kleene set

$$\overline{K} = \{x \mid W_{f(h(x))} \neq \emptyset\}$$

is semidecidable. We reach a contradiction. □

Problem 63. There is no computable function f such that if W_a is decidable, then $f(a)\downarrow$ and $f(a)$ is an index of the characteristic function for W_a, in other words, $\varphi^{(1)}_{f(a)} = \chi_{W_a}$.

✍ Homework!

Problem 64. There is no computable function f such that if $\varphi_a = \chi_A$ and A is finite, then $f(a)\downarrow$ and $A = D_{f(a)}$.

Here D_v denotes the finite set with code v.

Hint. Let $K = \{x \mid (\exists y)\kappa(x,y) = \texttt{True}\}$. Consider the total computable function

$$g(x,y) \overset{\text{def}}{=} \begin{cases} \texttt{True}, & \text{if } \kappa(x,y) = \texttt{True} \ \&\ (\forall z < y)[\kappa(x,z) = \texttt{False}] \\ \texttt{False}, & \text{otherwise.} \end{cases}$$

By the Parameters Theorem, there is a total computable function h such that $\varphi^{(1)}_{h(x)}(s) = g(x,s)$. Then

$$x \in \overline{K} \ \Leftrightarrow\ D_{f(h(x))} = \emptyset \ \Leftrightarrow\ f(h(x)) = 0.$$

It follows that $\overline{K}$ is semidecidable, which is a contradiction. □

Problem 65. There is no computable function ℓ such that if $\varphi_a^{(1)} = \chi_A$ and A is finite, then $\ell(a) = |A|$. ✍ Homework!

Problem 66. There is no computable function h such that if W_a is finite, then $h(a) \downarrow$ and $D_{h(a)} = W_a$.

Problem 67. Show that there exist primitive recursive functions α and β such that

$$W_{\alpha(a,b)} = \varphi_a^{-1}(W_b) \text{ and } W_{\beta(a,b)} = \varphi_a(W_b).$$

The next problem shows that we have to be careful when we take infinite unions and intersections.

Problem 68. Let A be semidecidable and B be decidable. Show that

1) $C = \bigcup_{e\in A} W_e$ is always semidecidable;

2) $\bigcup_{v\in B} D_v$ may not be decidable, only semidecidable; [4, p. 147, problem 33]

3) it is possible that neither $\bigcap_{e\in B} W_e$ nor its complement are semidecidable;

4) even if B is such that $(\forall e \in B)[W_e$ is decidable$]$, we may still have that neither $\bigcap_{e\in B} W_e$ nor its complement are semidecidable;

Proof.

1) This is easy. Firstly, let $A = W_a$, for some index a.

$$\begin{aligned} x \in C &\Leftrightarrow (\exists e \in A)[x \in W_e] && \text{// by def. of } C\\ &\Leftrightarrow (\exists e \in A)(\exists s)[T_1(e,x,s) = \texttt{True}] && \text{// by Normal Form Theorem}\\ &\Leftrightarrow (\exists e,s,t)[T_1(a,e,s) * T_1(e,x,s) = \texttt{True}] \end{aligned}$$

2) We will show that there exists a decidable set B such that $\bigcup_{v\in B} D_v$ is semidecidable, but not decidable.

   Since K is a semidecidable set, let κ a be primitive recursive predicate such that
   $$K = \{x \mid (\exists y)[\kappa(x,y) = \texttt{True}]\}.$$
   Define finite approximations of the set K in the following way:
   $$K_s \overset{\text{def}}{=} \{x \mid x < s \;\&\; (\exists t < s)[\kappa(x,t) = \texttt{True}]\}.$$
   Our first goal is to show that we can effective find the code for the finite set K, i.e. there is a total computable h such that $K_s = D_{h(s)}$.

Let $g(x,s) \overset{\text{def}}{=} \mathtt{sign}(\Sigma_{t<s}\kappa(x,t))$, which is obviously primitive recursive. Clearly,

$$K_s = \{x \mid x < s \;\&\; g(x,s) = \mathtt{True}\}.$$

Define the primitive recursive function

$$\begin{aligned} f(x,s) &= \begin{cases} 2^x, & \text{if } x \in K_s \\ 0, & \text{if } x \notin K_s \end{cases} \\ &= \begin{cases} 2^x, & \text{if } x < s \;\&\; g(x,s) = \mathtt{True} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Then define the primitive recursive function

$$h(s) = \sum_{x<s} f(x,s).$$

It is easy to see that $K_s = D_{h(s)}$. Since h is non-decreasing and takes arbitrarily large values, the set $B = \mathtt{rng}(h)$ will be computable because

$$\begin{aligned} x \in B &\implies (\exists s)[h(s) = x] \\ x \notin B &\implies (\exists s)[h(s) > x \;\&\; (\forall t < s)[h(t) \neq x]]. \end{aligned}$$

In the end,

$$K = \bigcup_s K_s = \bigcup_s D_{h(s)} = \bigcup_{v \in B} D_v.$$

3) By the Rice-Shapiro Theorem, we know that the sets `Inf` and `Fin` are not semidecidable. We will show that there exists a computable set B such that $\bigcap_{e\in B} W_e = \mathtt{Inf}$. Our construction of B will be based on the following observation:

$$\mathtt{Inf} = \{e \mid (\forall x)(\exists y > x)[\varphi_e(y)\downarrow]\}.$$

Consider the semidecidable set

Why is the set I semidecidable?

$$I = \{\langle x, e\rangle \mid (\exists y > x)[\varphi_e(y)\downarrow]\}.$$

Let $I = W_a$. By the Parameters Theorem, let h be a primitive recursive function such that

$$\varphi_a^{(1)}(\langle x, e\rangle) \simeq \varphi_{h(x)}^{(1)}(e),$$

and we know that h is strictly increasing. Then $B = \mathtt{rng}(h)$ is computable. We obtain the equalities

$$\bigcap_{e\in B} W_e = \bigcap_{x\in\mathbb{N}} W_{h(x)} = \{e \mid (\forall x)(\exists y > x)[\varphi_e(y)\downarrow]\} = \text{Inf}.$$

4) For every n, consider the decidable set

Why are the sets I_n decidable ?

$$I_n = \{x \mid x < n \;\&\; |W_x| \geq n\} \cup \{n, n+1, n+2, \dots\}.$$

Our proof is based on the observation that

$$\bigcap_n I_n = \text{Inf},$$

which is not a semidecidable set. Consider the semidecidable set

Why is the set I semidecidable ?

$$I = \{\langle n, x\rangle \mid x \geq n \;\vee\; |W_x| \geq n\}.$$

Fix an index e such that $W_e = I$. Then by the Parameters Theorem, there exists a primitive recursive h such that for every n and x,

$$\varphi_e^{(2)}(n, x) \simeq \varphi_{h(n)}^{(1)}(x).$$

Then we have that for every n, $W_{h(n)} = I_n$. Again, we can choose h so that it is strictly increasing. Let us consider the decidable set $B = \texttt{rng}(h)$. Clearly, $(\forall x \in B)[W_x$ is decidable$]$. We finish with the following observation:

This does **not** mean that $h(n)$ is an index of the characteristic function of the computable set I_n. It is an index of the semi-characteristic function

$$\bigcap_x W_x = \bigcap_n W_{h(n)} = \bigcap_n I_n = \text{Inf}.$$

□

Chapter 5

Effective Reducibilities

We already saw that many natural questions, such as whether a given program halts on every input, are undecidable and even not semidecidable. The most general way to prove this is by reducing a known undecidable (or non-semidecidable) question to the given question.

See the introduction of [1, Chapter 6].

- We say that the set A is **many-one reducible** to the set B, and write $A \leq_m B$, if there is a total computable function h such that

$$(\forall x)[x \in A \;\Leftrightarrow\; h(x) \in B].$$

Note that we may not have $h(A) = B$. We have $h^{-1}(B) = A$.

- We write $A \equiv_m B$ if $A \leq_m B$ and $B \leq_m A$.
- We say that a set A is **m-complete** if
  - A is semidecidable, and
  - for any semidecidable set W, we have $W \leq_m A$.
- We say that the set A is **one-one reducible** to the set B, and write $A \leq_1 B$, if there is a total computable *one-to-one* (injective) function h such that

$$(\forall x)[x \in A \;\Leftrightarrow\; h(x) \in B].$$

Here as well $h^{-1}(B) = A$.

- We write $A \equiv_1 B$ if $A \leq_1 B$ and $B \leq_1 A$.
- We write $A \equiv B$ if there is a total computable function h, which is also a *permutation* of $\mathbb{N}$, and

$$(\forall x)[x \in A \;\Leftrightarrow\; h(x) \in B].$$

In this case, $h(A) = B$ and $h^{-1}(B) = A$.

Example 9. In the proof of 4.3 we showed that

$$K \leq_m \overline{\texttt{Empty}}.$$

Proposition 5.1. Let f be a total computable function. The following are equivalent:

1) $(\forall x)[x \in A \Leftrightarrow f(x) \in B]$;

2) $A = f^{-1}(B)$;

3) $f(A) \subseteq B$ and $f(\overline{A}) \subseteq \overline{B}$.

[2, p. 159]

Proposition 5.2. Prove the following:

a) $A \leq_m B \Leftrightarrow \overline{A} \leq_m \overline{B}$;

b) if A is decidable set and $B \leq_m A$, then B is decidable;

c) if A is semidecidable and $B \leq_m A$, then B is semidecidable;

d) if A is decidable and $B \neq \emptyset, \mathbb{N}$, then $A \leq_m B$;

Proposition 5.3. Prove the following:

a) $A \leq_m \mathbb{N} \Leftrightarrow A = \mathbb{N}$;

b) $A \leq_m \emptyset \Leftrightarrow A = \emptyset$;

c) $\mathbb{N} \leq_m A \Leftrightarrow A \neq \emptyset$;

d) $\emptyset \leq_m A \Leftrightarrow A \neq \mathbb{N}$;

Problem 69. Prove the following:

- If A is a semidecidable set, then $A \leq_m \overline{A}$ iff A is decidable and $A \neq \emptyset, \mathbb{N}$.
- If A is semidecidable, but not decidable, then $A \not\equiv_m \overline{A}$.
- If the sets $A \setminus B$ and $B \setminus A$ are non-empty and finite, then $A \equiv_m B$.
- If $A \leq_m B$ via the function h and $\texttt{rng}(h) = \mathbb{N}$, then $B \leq_m A$.
- For any set A, $A \oplus \overline{A} \equiv_m \overline{A \oplus \overline{A}}$.
- There exists a non-semidecidable set A such that $A \equiv_m \overline{A}$. (Hint: consider $K \oplus \overline{K}$).

For

$$A \setminus B = \{a_0, \ldots, a_n\}$$
$$B \setminus A = \{b_0, \ldots, b_k\},$$

$$h(x) \overset{\text{def}}{=} \begin{cases} a_0, & \text{if } x \in B \setminus A \\ b_0, & \text{if } x \in A \setminus B \\ x, & \text{otherwise} \end{cases}$$

Problem 70. Suppose A and B are semidecidable sets such that $A \cup B = \mathbb{N}$ and $A \cap B \neq \emptyset$. Show that $A \leq_m A \cap B$.

Hint. Let $A = \bigcup_s A_s$ and $B = \bigcup_s B_s$. Fix $a_0 \in A \cap B$. Given x, the function h simultaneously checks if $x \in A_s$ or B_s, for $s = 0, 1, 2, \ldots$ If $x \in A_s$, $h(x) = a_0$. Otherwise, if $x \in B_s$, $h(x) = x$. We know that we will have one these two cases. □

Theorem 5.1. The set K is m-complete. Moreover, K is 1-complete.

Hint. Clearly, K is a semidecidable set. Consider another semidecidable set A. We will show that $A \leq_m K$. Let

$$f(x, y) \simeq \begin{cases} 42, & \text{if } x \in A \\ \uparrow, & \text{if } x \notin A. \end{cases}$$

By the Parameters Theorem, let h be total computable such that for every x and y,

We can take h to be one-to-one.

$$\varphi^{(1)}_{h(x)}(y) \simeq f(x, y).$$

Show that $A \leq_m K$ via the function h. □

We can apply the same proof to obtain the following corollary.

Corollary 5.1. The set $\overline{\texttt{Empty}}$ is 1-complete.

Corollary 5.2. If A is m-complete, B is semidecidable and $A \leq_m B$, then B is m-complete.

Corollary 5.3. For a set A, the following are equivalent:

(1) A is m-complete;

(2) $A \equiv_m K$;

(3) A is semidecidable and $K \leq_m A$.

5.1 The structure of many-one degrees

- $\equiv_m$ is an equivalence relation;

- $\mathtt{deg}_m(A) \overset{\text{def}}{=} \{B \mid A \equiv_m B\}$;

- $\boldsymbol{o} \overset{\text{def}}{=} \mathtt{deg}_m(\emptyset)$;

- $\boldsymbol{n} \overset{\text{def}}{=} \mathtt{deg}_m(\mathbb{N})$;

- $\mathbf{0}_m \overset{\text{def}}{=} \{A \mid A \text{ is decidable and } A \neq \emptyset, \mathbb{N}\}$;

- $\mathbf{0}'_m \overset{\text{def}}{=} \mathtt{deg}_m(K)$;

Proposition 5.4. Each pair of m-degrees has a least upper bound.

Hint. Let $\boldsymbol{a} = \mathtt{deg}_m(A)$ and $\boldsymbol{b} = \mathtt{deg}_m(B)$. Let

$$C = A \oplus B = \{2x \mid x \in A\} \cup \{2x + 1 \mid x \in B\}.$$

Show that $\boldsymbol{c} = \mathtt{deg}_m(C)$ is the least upper bound, i.e.

- $\mathbf{a}, \mathbf{b} \leq_m \mathbf{c}$;

- if $\boldsymbol{a}, \boldsymbol{b} \leq_m \boldsymbol{d}$, then $\boldsymbol{c} \leq_m \boldsymbol{d}$.

$\square$

- If we ignore $\mathbf{o}$ and $\mathbf{n}$, there is a minimal semidecidable degree, i.e. $\mathbf{0}_m$;

- There is a maximal semidecidable degree, i.e. $\mathbf{0}'_m$;

- The semidecidable m-degrees form an initial segment of the m-degrees.

Post's problem for m-degrees: Are there semidecidable sets which are neither decidable nor m-complete?

We will see that the simple sets are such.

5.2 The Myhill Isomorphism Theorem

Theorem 5.2 (Myhill Isomorphism Theorem 1955)**.** For any two sets of natural numbers A and B,

$$A \equiv B \quad \Leftrightarrow \quad A \equiv_1 B.$$

Proof. The direction ($\Rightarrow$) is immediate. For ($\Leftarrow$), let $A \leq_1 B$ by f and $B \leq_1 A$ by g. We will build a computable *permutation* h such that $h(A) = B$. At each step of the construction, we will build a finite injective h_s such that

[5, p. 325].

$$h = \bigcup_s h_s \ \& \ (\forall s)[h_s \subseteq h_{s+1}].$$

Let $h_0 = \emptyset$. Suppose we have built h_s. We will show how we build h_{s+1}.

- Let $s + 1 = 2x + 1$.
  - If $h_s(x) \downarrow$, then we do nothing.
  - If $h_s(x) \uparrow$, then we build a chain:

    $$x \xrightarrow{f} y_0 \xrightarrow{h_s^{-1}} x_1 \xrightarrow{f} y_1 \xrightarrow{h_s^{-1}} x_2 \xrightarrow{f} \cdots \xrightarrow{h_s^{-1}} x_n \xrightarrow{f} y_n,$$

    until we reach a number $y_n \notin \mathtt{rng}(h_s)$. Notice that all $x_i \in \mathtt{dom}(h_s)$.
  - Is it possible to build an *infinite* chain in this way? Since h_s is finite, this would mean that we have a cycle. Suppose we have a cycle. This means that there exist pairs (x_i, x_j) such that $x_i = x_j$ and $i < j$. Note that $x_i \neq x$, because $x \notin \mathtt{dom}(h_s)$, but $x_i \in \mathtt{dom}(h_s)$. Among all such pairs (x_i, x_j), consider the pair with the least index i. Let $x_0 = x$. Then

    If we assume that $y_i = y_j$, for $i < j$, then $f(x_i) = y_i = f(x_j)$ and hence $x_i = x_j$, because f is injective.

    $$h_s^{-1}(f(x_{i-1})) = x_i = x_j = h_s^{-1}(f(x_{j-1})).$$

    Since h_s^{-1} and f are injective, their composition is also injective, and hence $x_{i-1} = x_{j-1}$. A contradiction with the choice of the pair (x_i, x_j).
  - So, the chain is finite and we have a number $y_n \notin \mathtt{rng}(h_s)$. Define h_{s+1} such that $\mathtt{graph}(h_{s+1}) = \mathtt{graph}(h_s) \cup \{\langle x, y_n \rangle\}$. Moreover, for this

number x, we have the following chain of equivalences:

$$\begin{aligned}
x \in A &\Leftrightarrow f(x) = y \in B && \text{// since } A \leq_1 B \\
&\Leftrightarrow h_s^{-1}(y) = x_1 \in A && \text{// we follow the chain} \\
&\Leftrightarrow f(x_1) = y_1 \in B \\
&\ \vdots \\
&\Leftrightarrow f(x_n) = y_n \in B \\
&\Leftrightarrow h_{s+1}(x) = y_n \in B && \text{// by def. of } h_{s+1}.
\end{aligned}$$

We conclude that h_{s+1} is injective and

$$(\forall x \in \mathtt{dom}(h_{s+1}))[x \in A \ \Leftrightarrow\ h_{s+1}(x) \in B].$$

- Let $s + 1 = 2y + 2$.

This case is symmetrical.

  – If $h_s^{-1}(y)\downarrow$, then we do nothing.
  – If $h_s^{-1}(y)\uparrow$, then we build a chain:

$$y \xrightarrow{g} x \xrightarrow{h_s} y_1 \xrightarrow{g} x_1 \xrightarrow{h_s} y_2 \cdots \xrightarrow{g} x_n,$$

  until we reach a number $x_n \notin \mathtt{dom}(h_s)$. Notice that all $x_i \in \mathtt{rng}(h_s)$.
  – We define $\mathtt{graph}(h_{s+1}) = \mathtt{graph}(h_s) \cup \{\langle x_n, y\rangle\}$.

$\square$

Why is the produced function h computable ?

Problem 71. Let $K_0 = \{\langle e, x\rangle \mid \varphi_e(x)\downarrow\}$. Show that $K_0 \equiv K$.

Proof. First, we will show that $K_0 \leq_1 K$. Consider the computable function

$$f(u, x) \overset{\text{def}}{\simeq} \begin{cases} 5, & \text{if } \Phi_1(\lambda(u), \rho(u))\downarrow \\ \uparrow, & \text{otherwise.} \end{cases}$$

By the Parameters Theorem, there exists an one-to-one total computable function h such that $\varphi_{h(u)}(x) \simeq f(u, x)$.

$$\begin{aligned}
\langle e, x\rangle \in K_0 &\implies \varphi_e(x)\downarrow \implies \varphi_{h(\langle e,x\rangle)} \text{ is total} \implies h(\langle e, x\rangle) \in K \\
\langle e, x\rangle \notin K_0 &\implies \varphi_e(x)\uparrow \implies \varphi_{h(\langle e,x\rangle)} = \emptyset^{(1)} \implies h(\langle e, x\rangle) \notin K.
\end{aligned}$$

We conclude that $K_0 \leq_1 K$.

Second, consider the one-to-one function $h(x) = \langle x, x\rangle$. Clearly, $x \in K \Leftrightarrow h(x) \in K_0$. Hence, $K \leq_1 K_0$. $\square$

5.3 Productive and creative sets

Consider a set A that is not c.e. This means that for every index e such that $W_e \subset A$, there is a witness x of the fact that A is not equal to W_e, i.e. $x \in A \setminus W_e$. There is an interesting class of non-c.e. sets for which we can find such witnesses in an effective way. As a simple example, consider the set $\overline{K}$. For every $W_x \subset \overline{K}$, $x \in \overline{K} \setminus W_x$, i.e. the identity function $f(x) = x$ gives us the witness to the fact that $\overline{K}$ is not W_x.

- A set A is called **productive** if

$$(\exists e)(\forall x)[W_x \subset A \implies \varphi_e(x)\downarrow \;\&\; \varphi_e(x) \in A \setminus W_x].$$

As already noted, $\overline{K}$ is productive with productive function $f(x) = x$.

  The computable function φ_e with the above property will be called a **productive function** for the set A. Clearly, if a set A is productive, then it is not c.e.

Cutland [2] considers only *total* productive functions.

- A set C is **creative** if C is semidecidable and its complement $\overline{C}$ is a productive set.

Informally, a creative set C is "effectively non-decidable". Since C is semidecidable, to be computable means $\overline{C}$ to be semidecidable. Then for every possible candidate $W_x \subseteq \overline{C}$, we have an algorithm (the productive function π) for finding a witness to the fact that $\overline{C}$ is not semidecidable, i.e. $\pi(x) \in \overline{C} \setminus W_x$. Our goal here is to show that there are semidecidable sets which are not creative.

Proposition 5.5. If P is productive and $P \leq_m B$, then B is productive.

Proof. Let f be total computable such that $x \in P \iff f(x) \in B$, i.e. $f^{-1}(B) = P$, and let π be a productive function for P. By *Problem* 40, there exists a primitive recursive g such that $f^{-1}(W_x) = W_{g(x)}$, for every x. Consider $W_x \subset B$.

[2, p. 134]

$$W_{g(x)} = f^{-1}(W_x) \subset f^{-1}(B) = P.$$

Since $W_{g(x)} \subset P$ and π is a productive function for P, we have

$$\pi(g(x)) \in P \setminus W_{g(x)} \to \pi(g(x)) \in f^{-1}(B) \setminus f^{-1}(W_x) \to f(\pi(g(x))) \in B \setminus W_x.$$

We conclude that $f \circ \pi \circ g$ is a productive function for B. □

Theorem 5.3 (Post 1944)**.** Every productive set contains an infinite semidecidable set.

Proof. Let P be a productive set and π be a productive function for P. Clearly, $P \neq \emptyset$. Fix an index z_0 such that $W_{z_0} = \emptyset$. Then $\pi(z_0) \in P \setminus \emptyset = P$.

[8, p. 90], [5, p. 258], [2, p. 137].

Our goal is to build a total computable one-to-one function g such that $\texttt{rng}(g) \subset P$. Here is an idea how to do that:

$$\left| \begin{array}{l} g(0) = \pi(z_0) \\ g(n+1) = \pi(z_{n+1}), \quad \text{where } W_{z_{n+1}} = \{g(0), \dots, g(n)\} \subset P. \end{array} \right.$$

We have to show that we can define g following a primitive recursive scheme. More formally, we show that:

✍ **give a full proof!**

- there exists a primitive recursive f such that $W_{f(x,y)} = W_x \cup W_y$;
- there exists a primitive recursive h such that $W_{h(x)} = \{x\}$;
- there exists a computable function κ such that $\kappa(n) = z_n$. We define the function κ following the scheme:

$$\left| \begin{array}{lcl} \kappa(0) & = & z_0 \\ \kappa(n+1) & = & f(\kappa(n), h(\pi(\kappa(n)))). \end{array} \right.$$

In the end, we let $g = \pi \circ \kappa$. □

Corollary 5.4. The set $\overline{K}$ contains an infinite semidecidable set.

Lemma 5.1. If P is productive, then P has a *total* productive function.

Proof. Let π be a productive function for the set P. Consider the computable function with the following definition:

$$f(x,y) \overset{\text{def}}{\simeq} \begin{cases} \varphi_x(y), & \text{if } \pi(x)\downarrow \\ \uparrow, & \text{otherwise.} \end{cases}$$

By the Parameters Theorem, there is a primitive recursive g such that

$$W_{g(x)} = \begin{cases} W_x, & \text{if } \pi(x)\downarrow \\ \emptyset, & \text{otherwise.} \end{cases}$$

Then, for any number x, at least one of the computations $\pi(x)$ or $\pi(g(x))$ converges. Let $\hat{\pi}(x)$ be the return value of that computation which converges first.

The function $\hat{\pi}$ is productive for P because if $W_x \subseteq P$, then $\pi(x)\downarrow$ and hence $W_{g(x)} = W_x$. It follows that both $\pi(x) \in P \setminus W_x$ and $\pi(g(x)) \in P \setminus W_x$. Then $\hat{\pi}(x) \in P \setminus W_x$. □

Lemma 5.2. Every productive set P has a total *one-to-one* productive function.

Proof. Let π be a *total* productive function for P. Define the primitive recursive function h such that $W_{h(x)} = W_x \cup \{\pi(x)\}$. Clearly, we have

$$W_x \subset P \;\to\; W_x \subset W_{h(x)} \subset P \;\to\; W_{h(x)} \subset W_{h(h(x))} \subset P \;\to \cdots$$

We define the one-to-one computable function $\hat{\pi}$. Let $\hat{\pi}(0) \overset{\text{def}}{=} \pi(0)$. To define $\hat{\pi}(x+1)$, we start computing the sequence:

Recall $h^0 = id$, $h^{n+1} = h \circ h^n$

$$\pi(h^0(x+1)),\ \pi(h^1(x+1)),\ \pi(h^2(x+1)),\ \ldots \tag{5.1}$$

We can do that since π and h are total. We do this until:

✍ **give details!**

a) we find a number i_0 such that $\pi(h^{i_0}(x+1)) \notin \{\hat{\pi}(0), \hat{\pi}(1), \ldots, \hat{\pi}(x)\}$. In this case, we set
$$\hat{\pi}(x+1) \overset{\text{def}}{=} \pi(h^{i_0}(x+1)).$$

b) we find a repetition in the sequence (5.1). In this case, $W_{x+1} \not\subset P$ and hence it does not matter what the value of $\hat{\pi}(x+1)$ is. We let
$$\hat{\pi}(x+1) = \min\{y \in \mathbb{N} \mid y \notin \{\hat{\pi}(0), \hat{\pi}(1), \ldots, \hat{\pi}(x)\}\}.$$

□

Proposition 5.6. If P is productive, then $\overline{K} \leq_m P$. Even more, we can make sure that $\overline{K} \leq_1 P$.

Proof. Let π be a *total* productive function for P. We already know how to build a primitive recursive function $f(x, y)$ such that

✍ Show how we find f!

$$W_{f(x,y)} = \begin{cases} \{\pi(x)\}, & \text{if } y \in K \\ \emptyset, & \text{if } y \notin K. \end{cases}$$

By the Fixed point theorem with parameters, there exists a total computable function η such that

$$W_{\eta(y)} = W_{f(\eta(y),y)} = \begin{cases} \{\pi(\eta(y))\}, & \text{if } y \in K \\ \emptyset, & \text{if } y \notin K. \end{cases}$$

Since we have the following chains of implications,

$$\begin{aligned} y \in K &\rightarrow W_{\eta(y)} = \{\pi(\eta(y))\} \rightarrow W_{\eta(y)} \not\subset P \rightarrow \pi(\eta(y)) \notin P, \\ y \notin K &\rightarrow W_{\eta(y)} = \emptyset \rightarrow W_{\eta(y)} \subset P \rightarrow \pi(\eta(y)) \in P, \end{aligned}$$

we conclude that $\overline{K} \leq_m P$ by the total computable function $\pi \circ \eta$.

Since we can choose a *one-to-one* total productive function for P and we can take a *one-to-one* S^m_n function, we can show that $\overline{K} \leq_1 P$. □

Corollary 5.5. If C is creative, then $K \leq_1 C$.

We generalise everything we did until now in the following statement.

Theorem 5.4 (Myhill)**.** The following are equivalent:

(1) C is creative;

(2) C is m-complete;

(3) C is 1-complete;

(4) $C \equiv K$.

Proof.

(1) → (2) Let C be semidecidable and $\bar{C}$ be productive. By *Proposition* 5.6, $\overline{K} \leq_m \overline{C}$. Then $K \leq_m C$ and hence C is m-complete, since K is m-complete. The same argument can be applied to prove (1) → (3).

(2) → (1) Let C be m-complete. Thus, $K \leq_m C$ and $\overline{K} \leq_m \overline{C}$. Since $\overline{K}$ is productive, by *Proposition* 5.5, it follows that $\overline{C}$ is productive and hence C is creative. The same argument can be applied to prove (3) → (1).

(3) ↔ (4) Since K is 1-complete, then $C \leq_1 K$. The last corollary gives us $K \leq_1 C$. Then we apply *Theorem* 5.2.

□

5.4 Immune and simple sets

Here we will see that there exist m-degrees strictily between $\mathbf{0}_m$ and $\mathbf{0}'_m$.

- We know that there is a set $A \in \mathbf{0}'_m$ such that $\overline{A}$ contains an infinite semidecidable set.
- Clearly, there is a set $A \in \mathbf{0}_m$ such that $\overline{A}$ contains an infinite semidecidable set.
- It is natural to consider semidecidable sets whose complements does not contain infinite semidecidable sets.
- An infinite set I is called **immune** if it does not contain an infinite semidecidable set.
- A set S is called **simple** if
  - S is semidecidable;
  - $\overline{S}$ is infinite and immune.

Theorem 5.5 (Post 1944)**.** Simple sets exist.

Proof. Consider the semidecidable set

Here we follow [8, p. 106]. See also [5, p. 259]

$$C \overset{\text{def}}{=} \{\langle x, y\rangle \mid y \in W_x \;\&\; y > 2x\}.$$

By *Problem* 49, let h be a *total one-to-one computable* function such that

$$C = \texttt{rng}(h).$$

Define the partial order $\le_h$, where

$$\langle x, y\rangle \le_h \langle x', y'\rangle \quad\Leftrightarrow\quad (\exists m)(\exists n)[h(n) = \langle x, y\rangle \;\&\; h(m) = \langle x', y'\rangle \;\&\; n \le m].$$

Consider the semidecidable set

$$C' \overset{\text{def}}{=} \{\langle x, y\rangle \in C \mid (\forall z)[\langle x, z\rangle \in C \;\rightarrow\; \langle x, y\rangle \le_h \langle x, z\rangle]\}.$$

It is easy to see that there is a computable function ψ such that $\texttt{graph}(\psi) = C'$. Let $S = \texttt{rng}(\psi)$. We claim that S is a simple set.

- It is clear that S is a semidecidable set;

- We will show that $\mathbb{N} \setminus S$ is infinite. For every number x, there is *at most* one number y such that $\langle x, y\rangle \in C'$, and if such y exists, then $y > 2x$. In other words,

$$\texttt{rng}(\psi) \cap \{0, 1, \ldots, 2x\} \subseteq \texttt{rng}(\psi \restriction \{z \mid z < x\}).$$

  Thus,

$$|\{S \cap \{0, \ldots, 2x\}| \leq |\texttt{rng}(\psi \restriction \{z \mid z < x\})| \leq x.$$

  It follows that $\overline{S}$ is infinite.

- Let $W = W_b$ be an infinite semidecidable set. There are infinitely many numbers y such that $\langle b, y\rangle \in C$, i.e. $y \in W_b$ and $y > 2b$. Let y_0 be the least such y relative to $\leq_h$. Then, by construction, $y_0 \in S \cap W_b$. We conclude that $W_b \not\subseteq \overline{S}$.

□

Corollary 5.6. There exists a semidecidable set which is **not** creative.

Proof. Let S be a simple set. Assume S is creative. Then $\overline{S}$ will be productive. But by *Theorem* 5.3 there is an infinite semidecidable set $W \subseteq \overline{S}$. We reach a contradiction. □

Random numbers

- Let $K(x) \overset{\text{def}}{=} \mu e[\varphi_e(0) \simeq x]$. Here we follow [5, p. 261]

- $K(x)$ is called the Kolmogorov complexity of x.

- A number x is **random** if $x \leq K(x)$, i.e. the number x cannot be compressed. Clearly, the number 0 is random, according to this definition.

Proposition 5.7. There are infinitely many random numbers.

Hint. Let $k_0 \overset{\text{def}}{=} K(0)$. Consider the finite set

$$A_0 \overset{\text{def}}{=} \{y \mid (\exists e \leq k_0)[\varphi_e(0) \simeq y]\}.$$

Clearly, $\max(A_0) \leq k_0$ and every number *not* in A_0 will have complexity at least k_0+1. Let x_0 be the least number not in A_0. Then $x_0 \leq k_0+1 \leq K(x_0)$. We obtain a new random number $x_0 > 0$.

Now consider the set

$$A_1 \overset{\text{def}}{=} \{y \mid (\exists e \leq k_1)[\varphi_e(0) \simeq y]\}.$$

Again, $\max(A_1) \leq k_1$ and every number *not* in A_1 will have complexity greater that k_1. Let x_1 be the least number not in A_1. Then $x_1 \leq k_1 + 1 \leq K(x_1)$. We obtain a new random number $x_1 > x_0 > 0$.

Following this procedure, we can obtain an infinite sequence of random numbers. □

Proposition 5.8 (Kolmogorov)**.** There is no infinite semidecidable set of random numbers.

Hint. Suppose W is an infinite semidecidable set. We know that there exists a total computable f such that $W = \texttt{rng}(f)$. Consider the computable function

$$g(e, z) \overset{\text{def}}{=} f(\mu n[f(n) > e]).$$

In other words, we obtain the first enumerated by f element of W which is greater that e. Consider the total computable h such that $\varphi_{h(e)}(z) \simeq g(e, z)$. There is an index e such that $\varphi_e = \varphi_{h(e)}$. Then $\varphi_e(0) \simeq x > e$, for some element $x \in W$. It follows that $K(x) \leq e < x$ and hence x is a nonrandom number belonging to W. □

Theorem 5.6. The set of nonrandom numbers is simple.

Hint. Consider the set $S \overset{\text{def}}{=} \{x \mid K(x) < x\}$, the set of nonrandom numbers.

- Clearly, S is a semidecidable set.

- $\overline{S}$ is the set of random numbers and it is infinite by *Proposition* 5.7.

- Any $B \subseteq \overline{S}$ is a set of random numbers. We know that there is no semidecidable set of random numbers by *Proposition* 5.8.

$\square$

5.5 Problems

Problem 72. Let $A = \{a \mid W_a = \{a\}\}$. Show the following:

a) $K \leq_m A$;

b) $\overline{K} \leq_m A$.

Conclude that $K \oplus \overline{K} \leq_m A$.

Proof.

a) Consider the computable function

$$f(x, y, z) \overset{\text{def}}{\simeq} \begin{cases} 5, & \text{if } y \in K \;\&\; z = S^1_1(x, y) \\ \uparrow, & \text{otherwise.} \end{cases}$$

By the Recursion Theorem, there exists an index e such that $\varphi_e(y, z) \simeq f(e, y, z)$. Consider the total computable function $h(y) \overset{\text{def}}{=} S^1_1(e, y)$ such that

$$\varphi_{h(y)}(z) \simeq \varphi_e(y, z).$$

By following the chains of implications:

$$\begin{aligned} y \in K &\implies \mathtt{dom}(\varphi_{h(y)}) = \{h(y)\} \implies h(y) \in A \\ y \notin K &\implies \mathtt{dom}(\varphi_{h(y)}) = \emptyset \implies h(y) \notin A, \end{aligned}$$

we conclude that $K \leq_m A$.

b) Consider the computable function

$$f(x, y, z) \overset{\text{def}}{\simeq} \begin{cases} 5, & \text{if } x \in K \;\vee\; z = S^1_1(x, y) \\ \uparrow, & \text{otherwise.} \end{cases}$$

□

Problem 73. Suppose that f is a total injective computable function such that $\mathtt{rng}(f)$ is not decidable. Show that the set

$$A = \{x \mid (\exists y)[y > x \;\&\; f(y) < f(x)]\}$$

is simple.

Problem 74. Let $K_0 \overset{\text{def}}{=} \{\langle e, x\rangle \mid \varphi_e(x)\downarrow\}$. Show that

$$K \equiv_m K_0 \equiv_m \overline{\texttt{Empty}}.$$

Actually, here we can replace $\equiv_m$ by $\equiv_1$.

Problem 75. Show that

$$\texttt{Inf} \equiv_m \texttt{Tot} \equiv_m \texttt{Const}.$$

Here also we can replace $\equiv_m$ by $\equiv_1$.

Problem 76. Show the following:

a) $\overline{K} \leq_m \texttt{Tot}$;

b) $K \leq_m \texttt{Tot}$;

c) $\texttt{Tot} \not\leq_m \overline{K}$;

d) $\texttt{Tot} \not\leq_m K$;

e) Tot is productive.

Hint.

a) As usual, for the semidecidable set K, let κ be a primitive recursive function such that

We repeat an old argument here.

$$K = \{x \mid (\exists s)[\kappa(x,s) = \texttt{True}]\}.$$

Consider the computable function

$$f(x,y) \overset{\text{def}}{\simeq} \begin{cases} 42, & \text{if } (\forall s < y)[\kappa(x,s) = \texttt{False}] \\ \uparrow, & \text{if } (\exists s < y)[\kappa(x,s) = \texttt{True}]. \end{cases}$$

There is a primitive recursive g such that $\varphi_{g(x)}(y) \simeq f(x,y)$. Then

$$\begin{aligned} x \in K &\implies (\exists s)[\kappa(x,s) = \texttt{True}] \implies \varphi_{g(x)} \text{ is finite} \implies g(x) \notin \texttt{Tot}; \\ x \notin K &\implies (\forall s)[\kappa(x,s) = \texttt{False}] \implies \varphi_{g(x)} \text{ is total} \implies g(x) \in \texttt{Tot}. \end{aligned}$$

We conclude that $\overline{K} \leq_m \texttt{Tot}$.

b) Consider the computable function

$$f(x,y) \overset{\text{def}}{\simeq} \begin{cases} 42, & \text{if } x \in K \\ \uparrow, & \text{otherwise.} \end{cases}$$

There is a primitive recursive g such that $\varphi_{g(x)}(y) \simeq f(x,y)$. Then

$$\begin{aligned} x \in K &\implies \varphi_{g(x)} \text{ is total} \implies g(x) \in \texttt{Tot}; \\ x \notin K &\implies \varphi_{g(x)} = \emptyset^{(1)} \implies g(x) \notin \texttt{Tot}. \end{aligned}$$

We conclude that $K \leq_m \texttt{Tot}$.

c) Assume $\texttt{Tot} \leq_m \overline{K}$. Then $\overline{\texttt{Tot}} \leq_m K$. It follows that $\overline{\texttt{Tot}}$ is semidecidable, but by the Rice-Shapiro Theorem, $\overline{\texttt{Tot}}$ is not semidecidable. We conclude that $\texttt{Tot} \not\leq_m \overline{K}$.

d) Again, if we assume that $\texttt{Tot} \leq_m K$, then $\texttt{Tot}$ is semidecidable. By the Rice-Shapiro Theorem, $\texttt{Tot}$ is not semidecidable. We conclude that $\texttt{Tot} \not\leq_m K$.

e) We know that $\overline{K}$ is productive. Since $\texttt{Tot} \leq_m \overline{K}$, by *Proposition* 5.5, $\texttt{Tot}$ is also productive.

□

Problem 77. Let $\texttt{Ind}_x \overset{\text{def}}{=} \{y \mid \varphi_x = \varphi_y\}$.

a) Show that $\overline{K} \leq_m \texttt{Ind}_x$ for each index x. Hence, $\texttt{Ind}_x$ is productive for each index x.

b) Show that the reduction $\overline{K} \leq_m \texttt{Ind}_x$ is *not uniform* in x. This means that there is no total computable function $f(x,y)$ such that

$$(\forall y)[y \in \overline{K} \iff f(x,y) \in \texttt{Ind}_x].$$

Hint. As usual, for the semidecidable set K, let κ be a primitive recursive function such that [10, p. 43]

$$K = \{x \mid (\exists s)[\kappa(x,s) = \texttt{True}]\}.$$

a) We have two cases to consider.

- Suppose $\texttt{dom}(\varphi_x)$ is infinite. Consider the function

$$f(y,s) \overset{\text{def}}{\simeq} \begin{cases} \varphi_x(s), & \text{if } (\forall t \leq s)[\kappa(y,t) = \texttt{False}] \\ \uparrow, & \text{if } (\exists t \leq s)[\kappa(y,t) = \texttt{True}]. \end{cases}$$

Consider the total computable h such that $\varphi_{h(y)}(s) \simeq f(y,s)$.

$$y \in \overline{K} \implies \varphi_{h(y)} = \varphi_x \implies h(y) \in \mathtt{Ind}_x$$
$$y \notin \overline{K} \implies \varphi_{h(y)} \text{ is finite} \implies h(y) \notin \mathtt{Ind}_x.$$

We conclude that $\overline{K} \leq_m \mathtt{Ind}_x$.

- Suppose $\mathtt{dom}(\varphi_x)$ is finite. Consider the function

$$f(y,s) \overset{\text{def}}{\simeq} \begin{cases} \varphi_x(s), & \text{if } (\forall t \leq s)[\kappa(y,t) = \mathtt{False}] \\ 42, & \text{if } (\exists t \leq s)[\kappa(y,t) = \mathtt{True}]. \end{cases}$$

Consider the total computable h such that $\varphi_{h(y)}(s) \simeq f(y,s)$.

$$y \in \overline{K} \implies \varphi_{h(y)} = \varphi_x \implies h(y) \in \mathtt{Ind}_x$$
$$y \notin \overline{K} \implies \varphi_{h(y)} \text{ is infinite} \implies h(y) \notin \mathtt{Ind}_x.$$

We conclude that $\overline{K} \leq_m \mathtt{Ind}_x$.

b) Assume that f is computable and

$$(\forall y)[y \in \overline{K} \Leftrightarrow f(x,y) \in \mathtt{Ind}_x].$$

Fix some element $y_0 \in K$. Then the function $h(x) \overset{\text{def}}{=} f(x, y_0)$ is such that for all x, $h(x) \notin \mathtt{Ind}_x$, i.e. $\varphi_x \neq \varphi_{h(x)}$. But by the Fixed Point Theorem, there is an index e such that $\varphi_e = \varphi_{h(e)}$. It follows that $h(e) \in \mathtt{Ind}_e$. A contradiction.

□

Problem 78. Show that the following sets are m-equivalent to $\overline{K}$, where:

a) $\{x \mid W_x = \emptyset\}$;

b) $\{x \mid \varphi_x(5)\uparrow\}$;

c) $\{x \mid x \notin \mathtt{rng}(\varphi_x)\}$;

d) $\{x \mid \varphi_x(2x)\downarrow \implies \varphi_x(2x) \text{ is a prime number}\}$.

Problem 79. Show that K and $\overline{K}$ are m-reducible to the following sets:

a) $\{x \mid W_x = \{x\}\}$;

b) $\{x \mid n \notin W_x\}$, for a fixed number n;

c) $\{x \mid n \notin E_x\}$, for a fixed number n.

d) $\mathtt{Fin}$;

e) $\mathtt{Inf}$;

Bibliography

[1] S. Barry Cooper. *Computability Theory.* Chapman and Hall/CRC, 2003.

[2] Nigel Cutland. *Computability. An Introduction to Recursive Function Theory.* Cambridge University Press, 1980.

[3] Angel Ditchev and Ivan Soskov. *Theory of Programs (in Bulgarian).* Sofia University, 1996.

[4] Stela Nikolova and Alexandra Soskova. *Theory of Programs in Problems (in Bulgarian).* third. Softex, 2003.

[5] P. G. Odifreddi. *Classical Recursion Theory.* Vol. 125. Studies in logic and the foundations of mathematics. Elsevier, 1989.

[6] P. G. Odifreddi. *Classical Recursion Theory.* Vol. 143. Studies in logic and the foundations of mathematics. Elsevier, 1999.

[7] Rózsa Petér. *Recursive Functions.* third. Academic Press, 1967.

[8] Hartley Rogers Jr. *Theory of Recursive Functions and Effective Computability.* The MIT Press, 1987.

[9] John C. Shepherdson and Howard E. Sturgis. “Computability of Recursive Functions”. In: *Journal of ACM* 10.2 (1963), pp. 217 –255.

[10] Robert Soare. *Recursively Enumerable Sets and Degrees.* Perspectives of Mathematical Logic, Omega Series. Springer-Verlag, 1987.

Index