RECURSION AND COMPUTATION

YIANNIS N. MOSCHOVAKIS

Department of Mathematics
University of California, Los Angeles
and *University of Athens*

Version 1.2, December 2014

Recursion and computation
Version 1.0 was translated from the Greek by Garyfallia Vafeiadou
This is Version 1.2

Comments and corrections welcome, send to ynm@math.ucla.edu

CONTENTS

NOTATION

In these notes we will systematically use (as abbreviations) the following basic notations from logic and set theory:

$$\&: \text{and},\ \vee: \text{or},\ \neg: \text{not},\ \Longrightarrow: \text{implies},\ \Longleftrightarrow: \text{if and only if},$$
$$\forall: \text{for all},\ \exists: \text{there exists},\ \exists!: \text{there exists exactly one}$$

$$\begin{aligned}
x \in A &\iff \text{the element } x \text{ belongs to the set } A \\
A \subseteq B &\iff \text{every member of } A \text{ is a member of } B \\
&\iff (\forall x)[x \in A \Longrightarrow x \in B] \\
A = B &\iff \text{the sets } A \text{ and } B \text{ have exactly the same members} \\
&\iff A \subseteq B \ \&\ B \subseteq A \\
\{x \mid P(x)\} &= \text{the set of all } x \text{ which have property } P(x) \\
\{x \in A \mid P(x)\} &= \{x \mid x \in A \ \&\ P(x)\} \\
A \times B &= \{(a,b) \mid a \in A \text{ and } b \in B\} \\
&= \text{the set of ordered pairs } (a,b) \text{ with } a \in A, b \in B \\
A \times B \times C &= \{(a,b,c) \mid a \in A, b \in B, c \in C\} \\
&= \text{the set of triples } (a,b,c) \text{ with } a \in A, b \in B, c \in C
\end{aligned}$$

$$\begin{aligned}
f : A \to B &\iff f \text{ is a function with input set (domain) } A \\
&\qquad \text{and output set (range) } B \\
f : A \rightarrowtail B &\iff f \text{ is an injection (one-to-one function)} \\
f : A \twoheadrightarrow B &\iff f \text{ is a surjection (function onto } B) \\
f : A \rightarrowtail\!\!\!\!\twoheadrightarrow B &\iff f \text{ is a bijection (one-to-one and onto function)} \\
f : A \times B \to C &\iff f \text{ is a function of two variables, on } A \text{ and on } B
\end{aligned}$$

The best way to get used to these notations, if you are not familiar with them, is (in the beginning) to "translate" them and construct "paraphrases" of them in English. For example, the symbolic expression of the Induction Principle in section 1A.1 can be expressed in English as follows:

> Every set A of natural numbers has the following property: if A contains the number 0 and if, for every member n of A, the successor $n + 1$ is also a member of A, then all the natural numbers are members of A.

After some exercises of this kind the notation is learned and it becomes clear why its use is indispensable in mathematics.

We also note that in mathematical texts, we often use the same letter *in different alphabets or different fonts* to name different objects: so f is different from its Greek equivalent 'φ', and (even worse), in section 2B, systematically, 'x' names some "syntactic variable" which is assigned the natural number 'x'.

CHAPTER 1

PRIMITIVE AND μ-RECURSION

We introduce recursive definitions and we study two important classes of functions on the natural numbers, the *primitive recursive* and the μ-*recursive* functions.

1A. Recursive definitions and inductive proofs

The great 19th century mathematician Leopold Kronecker is alleged to have said that

God gave us the natural numbers, all the rest is the work of man.

In order to make use of the numbers, however, God also gave us the following fundamental

1A.1. **Induction Principle.** The characteristic property of the set of (natural) *numbers*

$$\mathbb{N} = \{0, 1, 2, \dots\}$$

is that *every set A that contains 0 and is closed under the successor operation $n \mapsto n+1$ contains all the numbers*, in symbols

$$\Big(0 \in A \ \&\ (\forall n)[n \in A \Longrightarrow n+1 \in A]\Big) \Longrightarrow \mathbb{N} \subseteq A.$$

Formally we appeal to the Principle of Induction in order to prove that all natural numbers have some property $P(n)$, by showing separately that

$$P(0) \text{ and } (\forall n)[P(n) \Longrightarrow P(n+1)].$$

From these propositions and the Induction Principle, it follows that the set $A = \{n \in \mathbb{N} \mid P(n)\}$ contains all the numbers, that is (for all n)$P(n)$.

The induction principle also justifies proofs by **complete induction**, in which we infer that all the natural numbers have some property $P(n)$ by showing that for every n,

$$(\forall i < n)P(i) \Longrightarrow P(n).$$

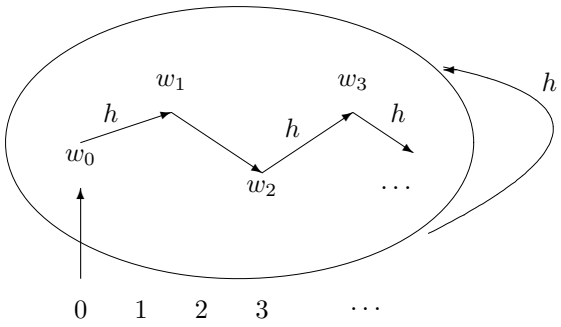


FIGURE 1. Recursive definition.

This is justified, because if we set

$$A = \{n \in \mathbb{N} \mid (\forall i < n)P(i)\},$$

then, obviously $0 \in A$, since there are no numbers $i < 0$) and so the proposition

$$(\forall i < 0)P(i)$$

is trivially true; and the required implication $n \in A \Longrightarrow n+1 \in A$ follows immediately from the induction hypothesis and the equivalence

$$(\forall i < n+1)P(i) \iff (\forall i < n)P(i) \text{ and } P(n).$$

The Induction Principle expresses our basic intuition that, if we start from 0 and repeat indefinitely the successor operation, then we will reach every natural number. The same intuition leads to the following fundamental result, which justifies **recursive definitions** on the set of natural numbers:

1A.2. **Basic Recursion Lemma.** *For all sets X, W and any given functions $g : X \to W$, $h : W \times \mathbb{N} \times X \to W$, there exists exactly one function $f : \mathbb{N} \times X \to W$ such that*

$$\begin{aligned} f(0,x) &= g(x), \\ f(n+1,x) &= h(f(n,x),n,x). \end{aligned} \tag{1}$$

In particular, without the parameter x, for every $w_0 \in W$ and every function $h : W \times \mathbb{N} \to W$, there exists exactly one function $f : \mathbb{N} \to W$ which satisfies the equations

$$f(0) = w_0, \quad f(n+1) = h(f(n),n). \tag{2}$$

Figure 1 illustrates a recursive definition in the simplest case, where the given function $h : W \to W$ does not depend on the *recursion variable* n or on some *parameter* x, that is when the function f is defined by the equations

$$f(0) = w_0, \quad f(n+1) = h(f(n)).$$

Usually the Basic Recursion Lemma is proved from the Induction Principle, which is why it is called a Lemma, cf. Problem x1A.1*. But the Induction Principle also follows from the Basic Recursion Lemma by Problem x1A.2*, and so these two principles express in different ways the same, characteristic property of the natural numbers.

1A.3. **Recursive definitions.** From the purely mathematical point of view, the Basic Recursion Lemma 1A.2 is a classical example of an *existence* and *uniqueness theorem* for the solution of a system of equations, (1), where "the unknown" is a function. Every proof of existence and uniqueness of an object with a specified property *defines* that object. The importance of the Basic Recursion Lemma flows from the following three fundamental properties of *recursive definitions*:

(I) *Most of the functions that arise in number theory and in computer science are defined—or can be defined—recursively from simpler functions.*

In the next section 1B we will get an idea of the richness of the set of "primitive recursive functions".

(II) The *recursive definition* (1) *produces a computable function f from given computable functions g and h.*

We will formulate this second principle rigorously and prove it in section 2B, but it is intuitively quite obvious: if we have "algorithms" which compute g and h, we can then compute any value $f(n, x)$ setting successively

$$\begin{aligned} f(0,x) &= g(x) && = w_0 \\ f(1,x) &= h(w_0, 0, x) && = w_1 \\ &\vdots \\ f(n-1,x) &= h(w_{n-2}, n-2, x) && = w_{n-1} \\ f(n,x) &= h(w_{n-1}, n-1, x). \end{aligned}$$

(III) *The form of recursive definition* (1) *leads in a natural way to inductive proofs of properties of the function $f(n, x)$.*

The connection

recursive definition – inductive proof

is one of the most fundamental in mathematics and theoretical computer science and we will investigate it in depth. Here we confine ourselves to two

examples, starting with the classical proof of the *commutativity of addition*, which uses the method of *double induction*. We write $f(x, y)$ instead of $x+y$ in this example, and we prove the commutativity of $f(x, y)$ using only its recursive definition.

1A.4. PROPOSITION. *The function $f(x, y)$ on $\mathbb{N}$ defined by the recursive equations*

$$f(0, y) = y$$
$$f(x + 1, y) = f(x, y) + 1$$

is commutative, that is, for all x, y

$$f(x, y) = f(y, x).$$

PROOF. We show by induction,

$$(\text{for all } x \in \mathbb{N})(\forall y)[f(x, y) = f(y, x)]. \tag{3}$$

BASIS, $x = 0$, $(\forall y)[f(0, y) = f(y, 0)]$. This is a proposition about all y and we will prove it by induction, the *Subsidiary Induction* for the basis of the "main" inductive proof of (3).

Subsidiary Basis, $y = 0$, $f(0, 0) = f(0, 0)$, obviously.

Subsidiary Inductive Step. We accept the *Subsidiary Induction Hypothesis*

$$f(0, y) = f(y, 0) \tag{SIH}$$

and we derive from it

$$f(0, y + 1) = f(y + 1, 0)$$

by a simple computation:

$$\begin{aligned} f(y + 1, 0) &= f(y, 0) + 1 && \text{(Definition)} \\ &= f(0, y) + 1 && \text{(SIH)} \\ &= y + 1 && \text{(Definition)} \\ &= f(0, y + 1) && \text{(Definition).} \end{aligned}$$

At this point we have completed the Subsidiary Induction and proved the BASIS of the main induction.

INDUCTIVE STEP. We accept the INDUCTION HYPOTHESIS

$$(\forall y)[f(x, y) = f(y, x)] \tag{IH}$$

and we show, with another Subsidiary Induction, that

$$(\forall y)[f(x + 1, y) = f(y, x + 1)].$$

Subsidiary Basis, $f(x + 1, 0) = f(0, x + 1)$. This follows from the proof of the BASIS, where we showed that for every y, $f(y, 0) = f(0, y)$.

Subsidiary Inductive step. We accept the *Subsidiary Induction Hypothesis*

(SIH) $$f(x+1,y) = f(y,x+1)$$

and we verify

$$f(x+1,y+1) = f(y+1,x+1)$$

by the following computation:

$$\begin{aligned} f(x+1,y+1) &= f(x,y+1)+1 && \text{(Definition)}\\ &= f(y+1,x)+1 && \text{(IH)}\\ &= (f(y,x)+1)+1 && \text{(Definition)}\\ &= (f(x,y)+1)+1 && \text{(IH)}\\ &= f(x+1,y)+1 && \text{(Definition)}\\ &= f(y,x+1)+1 && \text{(SIH)}\\ &= f(y+1,x+1) && \text{(Definition).} \end{aligned}$$ ⊣

1A.5. Remark. This method of proof is called *double induction*, because the Basis and the Induction Step of the "main" induction are also proved inductively. It is a feature of inductive proofs of propositions of the form

$$(\forall x)(\forall y)P(x,y)$$

like (3). The need for it becomes obvious if we try to prove directly the special case

$$(\forall x)[f(x,17) = f(17,x)].$$

Caution: the Induction Principle can be used to prove propositions of the form

(4) $$(\text{for all } n \in \mathbb{N})P(n),$$

and only propositions of this form. For example, if we want to prove by induction some proposition of the form

$$(\forall n)(\forall m)Q(n,m),$$

we need to choose which $P(n)$ we will use, e.g.,

$$\begin{aligned} P(n) &\iff Q(n,y) \quad (\text{for fixed, constant } y),\\ P(n) &\iff Q(x,n) \quad (\text{for fixed, constant } x),\\ P(n) &\iff (\forall y)Q(n,y),\\ P(n) &\iff (\forall x)Q(x,n), \end{aligned}$$

or even some more complex proposition $(\forall n)P(n)$ for which we can prove independently that

$$(\forall n)P(n) \Longrightarrow (\forall x)(\forall y)Q(x,y).$$

In some cases, the most difficult part of an inductive proof is the choice of some proposition of the form (4)—the *induction loading device*—which is easy to show and implies the proposition we are interested in.

As a second example we consider a function less familiar than addition but with equally interesting properties and many applications.

1A.6. The **Ackermann function** is defined by the so-called *double recursion*

$$\left\{\begin{array}{rcl} A(0,x) &=& x+1 \\ A(n+1,0) &=& A(n,1) \\ A(n+1,x+1) &=& A(n,A(n+1,x)); \end{array}\right. \tag{5}$$

and for every n, the *section* $A_n : \mathbb{N} \to \mathbb{N}$ of the Ackermann is

$$A_n(x) = A(n,x). \tag{6}$$

For example,

$$A_0(x) = x+1,$$

that is A_0 is the *successor* function S on the natural numbers.

Definition (5) must be justified and its justification is interesting, because it requires an appeal to the Basic Recursion Lemma for the definition of a function $f : \mathbb{N} \to W$ where W is a set substantially more complex than $\mathbb{N}$.

1A.7. LEMMA. *The system of functional equations* (5) *has exactly one solution,* that is, it is satisfied by exactly one two-place function.

PROOF. Let W be the set of all one-place functions on the natural numbers, that is

$$p \in W \iff p \text{ is a function, } p : \mathbb{N} \to \mathbb{N}.$$

We define a function $f : \mathbb{N} \to W$ by appealing to the Basic Recursion Lemma, as follows:

$$f(0) = S,$$

that is the value $f(0)$ is the successor function, $S(x) = x+1$; and

$$f(n+1) = h(f(n)),$$

where the value $g_p = h(p)$ of the function $h : W \to W$ is defined for every $p : \mathbb{N} \to \mathbb{N}$ by the recursion

$$g_p(0) = p(1), \quad g_p(x+1) = p(g_p(x)).$$

If we set

$$A_n = f(n),$$

then the functions A_n satisfy the following equations:

$$A_0(x) = S(x) = x+1, \quad A_{n+1}(x) = h(A_n)(x),$$

so that

$$A_{n+1}(0) = h(A_n)(0) = A_n(1),$$
$$A_{n+1}(x+1) = A_n(h(A_n)(x)) = A_n(A_{n+1}(x)).$$

Finally we set

$$A(n,x) = A_n(x)$$

and rewrite these equations,

$$A(0,x) = A_0(x) = x+1$$
$$A(n+1,0) = A_{n+1}(0) = A_n(1) = A(n,1)$$
$$A(n+1,x+1) = A_{n+1}(x+1) = A_n(A_{n+1}(x)) = A(n,A(n+1,x)).$$

These are exactly the equations for which we wanted to show that they have a solution.

The uniqueness of the solution is shown by double induction on n, or by a careful application of the Basic Lemma, which guarantees the uniqueness of the function $f(n) = A_n$, Problem x1A.7. ⊣

The Induction Principle and the Basic Recursion Lemma are fundamental axioms for the natural numbers that cannot be proved, unless we have some particular definition of the numbers in the context of a more general theory, e.g., the theory of sets.

Problems for Section 1A

x1A.1*. Prove the Basic Recursion Lemma 1A.2 from the Induction Principle. Hint: To define some $f : \mathbb{N} \times X \to W$ which satisfies the given equations, we set

$$W^n = \{(w_0,\dots,w_{n-1}) \mid w_0,\dots,w_{n-1} \in W\} \quad (n \in \mathbb{N}),$$
$$P(n,x,w) \iff w = (w_0,\dots,w_n) \in W^{n+1}$$
$$\& \; w_0 = g(x) \; \& \; (\forall i < n)[w_{i+1} = h(w_i,i,x)]$$

and we show by induction the proposition

$$(\forall n)(\exists! w)P(n,x,w).$$

Now for any n and x, let $w = w(n,x) = (w_0(n,x),\dots,w_n(n,x))$ be the unique sequence of length $(n+1)$ for which $P(n,x,w)$ holds and set

$$f(n,x) = s \iff s = w_n(n,x).$$

x1A.2*. Assume the Basic Recursion Lemma 1A.2 as an axiom and prove the Induction Principle.

x1A.3. Prove that every non-empty set of natural numbers $X \subseteq \mathbb{N}$ has a least element.

x1A.4. Prove that for any pair of real numbers $\alpha \geq 0$, $\beta > 0$, there exists exactly one natural number q, such that for some (real) r,

$$\alpha = \beta q + r, \quad 0 \leq r < \beta.$$

It follows that r is also unique, since $r = \alpha - \beta q$. The numbers q and r are the *quotient* and the *remainder* of the division of α by β, and we denote them by

$$\mathrm{quot}(\alpha, \beta) = q, \quad \mathrm{rem}(\alpha, \beta) = r.$$

It is also convenient to set

$$\mathrm{quot}(\alpha, 0) = 0, \quad \mathrm{rem}(\alpha, 0) = \alpha,$$

so that these functions are defined for all α, β and always satisfy the equation $\alpha = \beta \cdot \mathrm{quot}(\alpha, \beta) + \mathrm{rem}(\alpha, \beta)$.

x1A.5. Justify recursive definitions of the form

$$\begin{aligned} f(0, x) &= g_1(x), \\ f(1, x) &= g_2(x), \\ f(n+2, x) &= h(f(n, x), f(n+1, x), n, x), \end{aligned} \tag{7}$$

where g_1, g_2, h are given functions on the numbers.

x1A.6. The Fibonacci sequence is defined by the recursion

$$a_0 = 0, \quad a_1 = 1, \quad a_{n+2} = a_n + a_{n+1}. \tag{8}$$

(1) Compute the value a_9.
(2) Prove that for every n,

$$a_{n+2} \geq \lambda^n, \text{ where } \lambda = \frac{1 + \sqrt{5}}{2}.$$

The basic observation is that λ is one of the roots of the second order equation

$$x^2 = x + 1. \tag{9}$$

Prove also that if $\rho = \frac{1-\sqrt{5}}{2}$ is the other root of (9), then, for every n

$$a_n = \frac{\lambda^n - \rho^n}{\sqrt{5}}.$$

x1A.7. Prove that at most one function satisfies the system (5).

x1A.8. Compute the value $A(3, 2)$.

x1A.9. For the Ackermann sections, show that

$$A_1(x) = x + 2, \quad A_2(x) = 2x + 3.$$

x1A.10. Find a "closed" formula for $A_3(x)$, like those for A_1 and A_2 in the preceding problem.

x1A.11. Prove that for every n and every x, $A_n(x) \geq 1$.

x1A.12. Prove that every section A_n of the Ackermann function is *strictly increasing*, that is

$$x < y \Longrightarrow A_n(x) < A_n(y).$$

Infer that for all n, x, $A_n(x) \geq x$. Hint: Prove by double induction that $A_n(x) < A_n(x+1)$.

x1A.13. Prove that for all n, m and x,

$$n < m \Longrightarrow A_n(x) < A_m(x).$$

Hint: Prove by double induction that $A_n(x) < A_{n+1}(x)$.

x1A.14. Prove that for all n and x,

$$A_n(A_n(x)) < A_{n+2}(x).$$

1B. Primitive recursive functions

1B.1. Definition. A set F of functions of several variables[1] on the natural numbers is **primitively closed** if:

(1) The *successor* function $S(x) = x + 1$ belongs to F.

(2) For every n and q, the constant function of n variables

$$C_q^n(x_1, \ldots, x_n) = q$$

belongs to F. If $n = 0$, then (by convention) $C_q^0 = q$, that is we identify a function of "0 variables" with its (unique) value.

(3) For every n and i, $1 \leq i \leq n$, the *projection*

$$P_i^n(x_1, \ldots, x_n) = x_i$$

belongs to F. Notice that P_1^1 is the *identity* function on $\mathbb{N}$, $P_1^1(x) = x$.

(4) **Closure under composition**. If the m-place $g(u_1, \ldots, u_m)$ and the m, n-place functions

$$h_1(\vec{x}), \ldots, h_m(\vec{x})$$

belong to F, with $\vec{x} = (x_1, \ldots, x_n)$, then

$$f(\vec{x}) = g(h_1(\vec{x}), \ldots, h_m(\vec{x})) \tag{10}$$

[1] A function **of several variables** on the set M is any function

$$f : M^n \to M$$

of one or more variables on M, and the **arity** of f is the number n of its variables. The simultaneous study of all functions of several variables on a given set distinguishes logic and computation theory from most other branches of mathematics where, typically, we study separately the functions of one, or two, ... , or n variables.

also belongs to F.

(5) **Closure under primitive recursion**. If the n-place g and the $(n+2)$-place h belong to F, and if the $(n+1)$-place f is defined by the equations

$$\begin{cases} f(0,\vec{x}) = g(\vec{x}) \\ f(y+1,\vec{x}) = h(f(y,\vec{x}),y,\vec{x}), \end{cases} \tag{11}$$

then f also belongs to F. By convention, we include in this scheme the case $n=0$, where the scheme takes the form

$$\begin{cases} f(0) = q & (= C_q^0) \\ f(y+1) = h(f(y),y). \end{cases}$$

A function f is **primitive recursive** if it belongs to every primitively closed set of functions. More generally, for any set Ψ of functions of several variables, a function f is **primitive recursive in** Ψ if it belongs to every primitively closed set which contains Ψ. We use the notations:

$$\mathcal{R}_p = \{f \mid f \text{ is primitive recursive}\},$$
$$\mathcal{R}_p(\Psi) = \{f \mid f \text{ is primitive recursive in } \Psi\},$$

so that

$$\mathcal{R}_p = \mathcal{R}_p(\emptyset).$$

For example, addition

$$s(x,y) = x+y$$

is primitive recursive because it satisfies the equations

$$\begin{aligned} s(0,y) &= P_1^1(y) &&= y, \\ s(x+1,y) &= h(s(x,y),x,y) &&= s(x,y)+1, \end{aligned} \tag{12}$$

where the function $h(w,x,y) = S(w)$ is primitive recursive because it is defined by the composition

$$h(w,x,y) = S(P_1^3(w,x,y)). \tag{13}$$

With similar use of projections we can show that the set $\mathcal{R}_p(\Psi)$ is closed under very general explicit definitions. For example, if

$$f(x,y) = h(x, g_1(y,x+1), g_2(y,y)),$$

then f is primitive recursive in h, g_1, g_2, because

$$f(x,y) = h(P_1^2(x,y), g_1^*(x,y), g_2^*(x,y)), \tag{14}$$

where

$$S^*(x,y) = S(P_1^2(x,y)) \qquad = x+1 \tag{15}$$
$$g_1^*(x,y) = g_1(P_2^2(x,y), S^*(x,y)) \qquad = g_1(y,x+1) \tag{16}$$
$$g_2^*(x,y) = g_2(P_2^2(x,y), P_2^2(x,y)) \tag{17}$$

so that f is defined from the given functions by successive applications of composition.

The following proposition is trivial but useful:

1B.2. Proposition. *The set $\mathcal{R}_p(\Psi)$ of functions which are primitive recursive in Ψ contains Ψ and is primitively closed.*

Proof. If, e.g., f is defined by primitive recursion from $g, h \in \mathcal{R}_p(\Psi)$, then, $g, h \in F$ for every primitively closed F which contains Ψ; so, $f \in F$, for every primitively closed F which contains Ψ; so $f \in \mathcal{R}_p(\Psi)$. ⊣

For another example, the functions

$$\begin{aligned} g(w,y) &= w + y \\ h(w,x,y) &= g(P^3_1(w,x,y), P^3_3(w,x,y)) = w + y \end{aligned}$$

are primitive recursive, and therefore, if we set

$$\begin{aligned} f(0,y) &= 0 && = C^1_0(y) \\ f(x+1,y) &= h(f(x,y),x,y) && = f(x,y) + y, \end{aligned} \tag{18}$$

then $f(x,y)$ is also primitive recursive; but, obviously (by induction on x, if it does not seem obvious!), $f(x,y) = x \cdot y$, so multiplication too is a primitive recursive function.

In the future we will sometimes apply Proposition 1B.2 tacitly, without explicit mention.

1B.3. Proposition. *Addition $x+y$, multiplication $x \cdot y$ and the following functions are primitive recursive.*

#1. Factorial:
$x! = 1 \cdot 2 \cdots x$

$$\begin{aligned} 0! &= 1 \\ (x+1)! &= x!(x+1) \end{aligned}$$

#2. Predecessor:
$Pd(x) = \text{if } (x = 0) \text{ then } 0 \text{ else } x - 1$

$$\begin{aligned} Pd(0) &= 0 \\ Pd(x+1) &= x \end{aligned}$$

#3. Arithmetic subtraction:
$x \dot{-} y = \text{if } (x < y) \text{ then } 0 \text{ else } x - y$

$$\begin{aligned} x \dot{-} 0 &= x \\ x \dot{-} (y+1) &= Pd(x \dot{-} y) \end{aligned}$$

#4. $\min(x,y)$ $\quad \min(x,y) = x \dot{-} (x \dot{-} y)$

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

#6. $|x-y|$ $\quad |x-y| = (x \dot{-} y) + (y \dot{-} x)$

#7. x^y

$$\begin{aligned} x^0 &= 1 \\ x^{y+1} &= x^y \cdot x \end{aligned}$$

The proof of this and several more of the results in this section are easy, and we will skip them or deal with them in the problems.

1B.4. DEFINITION. The **characteristic function** of a relation $P(\vec{x})$ is the function

$$\chi_P(\vec{x}) = \begin{cases} 1, & \text{if } P(\vec{x}), \\ 0, & \text{otherwise,} \end{cases} \tag{19}$$

and the relation $P(\vec{x})$ is **primitive recursive** if $\chi_P(\vec{x})$ is primitive recursive. The same for sets: the characteristic function of $A \subseteq \mathbb{N}$ is the function

$$\chi_A(x) = \begin{cases} 1, & \text{if } x \in A, \\ 0, & \text{otherwise,} \end{cases} \tag{20}$$

and A is primitive recursive if χ_A is primitive recursive.

1B.5. PROPOSITION (Definition by cases). *If $P(\vec{x})$ is a primitive recursive relation, $g(\vec{x})$ and $h(\vec{x})$ are primitive recursive functions and $f(\vec{x})$ is defined from them by cases,*

$$f(\vec{x}) = \begin{cases} g(\vec{x}), & \text{if } P(\vec{x}), \\ h(\vec{x}), & \text{otherwise,} \end{cases}$$

then $f(\vec{x})$ is also primitive recursive.

PROOF. $f(\vec{x}) = \chi_P(\vec{x})g(\vec{x}) + (1 \dot{-} \chi_P(\vec{x}))h(\vec{x})$. ⊣

By applying this Proposition repeatedly, we can show that the set of primitive recursive functions is closed under definitions with n cases, for every $n \geq 2$.

1B.6. PROPOSITION. (1) *The following functions and relations are primitive recursive*:

#8. $x = y$

$$\chi_{=}(x, y) = 1 \dot{-} |x - y|$$

#9. $x \leq y$, $x < y$

$$\begin{aligned} \chi_{\leq}(x, y) &= 1 \dot{-} (x \dot{-} y) \\ \chi_{<}(x, y) &= \chi_{\leq}(x + 1, y) \end{aligned}$$

#10. $\mathrm{rem}(x,y)$ $\quad \mathrm{rem}(0,y) = 0$

$$\mathrm{rem}(x+1,y) = \begin{cases} x+1, \text{ if } y = 0, \\ \mathrm{rem}(x,y)+1, \\ \quad \text{otherwise, if } \mathrm{rem}(x,y)+1 < y, \\ 0, \text{otherwise} \end{cases}$$

#11. $x \mid y$ $\quad$ (x divides y) $\quad \chi_{|}(x,y) = 1 \dot{-} \mathrm{rem}(y,x)$

#12. $\mathrm{quot}(x,y)$ $\quad \mathrm{quot}(0,y) = 0$

$$\mathrm{quot}(x+1,y) = \begin{cases} 0, \text{ if } y = 0, \\ \mathrm{quot}(x,y)+1, \\ \quad \text{otherwise, if } \mathrm{rem}(x+1,y) = 0, \\ \mathrm{quot}(x,y), \text{otherwise} \end{cases}$$

(2) *The set of primitive recursive relations is closed under the propositional operators* $\neg, \vee, \ \& \ , \implies$, e.g., if

$$P(\vec{x}) \iff Q(\vec{x}) \ \& \ R(\vec{x})$$

and Q, R are primitive recursive, then P is also primitive recursive.

(3) *If the relation* $Q(\vec{y})$ *and the functions* $f_1(\vec{x}), \dots, f_m(\vec{x})$ *are primitive recursive, then the relation*

$$P(\vec{x}) \iff Q(f_1(\vec{x}), \dots, f_m(\vec{x}))$$

is also primitive recursive.

(4) *If* $g(i,\vec{x})$ *is primitive recursive, then so are the functions*

$$f(y,\vec{x}) = \textstyle\sum_{i<y} g(i,\vec{x}), \quad g(y,\vec{x}) = \prod_{i<y} g(i,\vec{x}),$$

with

$$\textstyle\sum_{i<0} g(i,\vec{x}) = 0, \quad \prod_{i<0} g(i,\vec{x}) = 1.$$

(5) *If* $P(i,\vec{x})$ *is primitive recursive and*

$$Q(z,\vec{x}) \iff (\exists i \le z) P(i,\vec{x})$$
$$R(z,\vec{x}) \iff (\forall i \le z) P(i,\vec{x}),$$

then $Q(z,\vec{x})$, $R(z,\vec{x})$ *are also primitive recursive.*

For the proofs we refer again to the problems.

1B.7. Corolary. *If the relation* $P(i,\vec{x})$ *and the function* $f(\vec{x})$ *are primitive recursive, then the relations*

$$Q(\vec{x}) \iff (\exists i \le f(\vec{x})) P(i,\vec{x})$$
$$R(\vec{x}) \iff (\forall i \le f(\vec{x})) P(i,\vec{x}),$$

are also primitive recursive, and the same with $<$ in the place of $\leq$.

It follows that the primality relation

$$\text{Prime}(x) \iff x \text{ is a prime number}$$

is primitive recursive.

PROOF in Problem x1B.10. ⊣

1B.8. The **bounded minimalization** operator is defined by

$$(\mu i \leq z)R(i,\vec{x}) = \begin{cases} \text{the least } i \leq z \text{ such that } R(i,\vec{x}), & \text{if } (\exists i \leq z)R(i,\vec{x}), \\ z+1 & \text{otherwise.} \end{cases}$$

1B.9. PROPOSITION. *For every primitive recursive relation $R(i,\vec{x})$, the function*

$$f(z,\vec{x}) = (\mu i \leq z)R(i,\vec{x})$$

is primitive recursive. It follows that if $g(\vec{x})$ is also primitive recursive, then the function

$$h(\vec{x}) = (\mu i \leq g(\vec{x}))R(i,\vec{x}) \quad (= f(g(\vec{x}),\vec{x}))$$

is primitive recursive.

PROOF in Problem x1B.12. ⊣

1B.10. COROLARY. *The function*

$$p_i = \text{the } i\text{'th prime number}$$

is primitive recursive.

PROOF. The function p_i is defined by primitive recursion

$$\begin{aligned} p_0 &= 2 \\ p_{i+1} &= (\mu t \leq p_i! + 1)[p_i < t \ \&\ \text{Prime}(t)], \end{aligned}$$

because (easily, x1B.11) for every k, there exists a prime number p such that $k < p \leq k! + 1$. ⊣

1B.11. **Codings**. A *coding* of a set A in a set C is any injection

$$c : A \rightarrowtail C,$$

which (theoretically) allows us to "recover" any element $x \in A$ from its *code* $c(x)$, for example, the function which assigns to every adult US citizen their Social Security number. Coding is a basic technique of logic and computability theory, characteristic of the subjects, and we will define several codings of many sets, with various, useful properties.

1B.12. **Sequence codings**. Let $\mathbb{N}^*$ be the set of all *finite sequences* of natural numbers, so that $\Lambda, (0), (1,5), (2,7,3), \ldots \in \mathbb{N}^*$, where Λ is the empty sequence. A coding

$$\langle\ \rangle : \mathbb{N}^* \rightarrowtail \mathbb{N}$$

of $\mathbb{N}^*$ in $\mathbb{N}$ is **primitive recursive**, if for every sequence of natural numbers $(u_0, \ldots, u_{n-1})$,

$$u_i < \langle u_0, \ldots, u_{n-1}\rangle \quad (i < n); \tag{21}$$

the relation

$$\mathrm{Seq}(u) \iff u = \langle u_0, \ldots, u_{n-1}\rangle \text{ for some } u_0, \ldots, u_{n-1} \tag{22}$$

is primitive recursive; for every n, the n-place function

$$f_n(u_0, \ldots, u_{n-1}) = \langle u_0, \ldots, u_{n-1}\rangle \tag{23}$$

is primitive recursive; and there are primitive recursive functions which satisfy the following:

$$\begin{aligned} &\mathrm{lh}(\langle u_0, \ldots, u_{n-1}\rangle) = n \\ &\mathrm{proj}(\langle u_0, \ldots, u_{n-1}\rangle, i) = (\langle u_0, \ldots, u_{n-1}\rangle)_i = u_i \qquad (i < n) \\ &\mathrm{append}(\langle u_0, \ldots, u_{n-1}\rangle, y) = \langle u_0, \ldots, u_{n-1}, y\rangle. \end{aligned} \tag{24}$$

It is also technically useful to require that

$$[\neg\mathrm{Seq}(u) \vee i \geq \mathrm{lh}(u)] \Longrightarrow \mathrm{lh}(u) = (u)_i = 0,$$

although the values $\mathrm{lh}(u)$, $(u)_i$ are of no importance when u is not a sequence code or i is greater than the length of the sequence coded by u. We observe that with (21), these requirements imply that for all u,

$$u > 0 \Longrightarrow (u)_i < u. \tag{25}$$

1B.13. PROPOSITION. *There exists a primitive recursive coding of* $\mathbb{N}^*$, *specifically the "classical" coding*

$$\langle u_0, \ldots, u_{n-1}\rangle = p_0^{u_0+1} \cdot p_1^{u_1+1} \cdots p_{n-1}^{u_{n-1}+1} \tag{26}$$

with $\langle\Lambda\rangle = 1$.

PROOF in Problem x1B.15. ⊣

The classical coding of $\mathbb{N}^*$ is not "efficient", and we will introduce in the problems more realistic codings which are used in *complexity* studies. From the point of view of *computability*, however, which is our main concern, all primitive recursive codings of $\mathbb{N}^*$ are equivalent, cf. Problem x1B.25.

From now on we fix a specific primitive recursive coding $\langle\ \rangle : \mathbb{N}^* \rightarrowtail \mathbb{N}$, which need not be the classical one.

1B.14. PROPOSITION. *There exist primitive recursive functions $u \restriction i$* (restriction) *and $u * v$* (concatenation)*, such that*

$$\langle u_0, \dots, u_{n-1}\rangle * \langle v_0, \dots, v_{m-1}\rangle = \langle u_0, \dots, u_{n-1}, v_0, \dots, v_{m-1}\rangle$$
$$\langle u_0, \dots, u_{n-1}\rangle \restriction i = \langle u_0, \dots, u_{i-1}\rangle \qquad (i \leq n).$$

PROOF in Problem x1B.16. ⊣

1B.15. PROPOSITION (**Mutual primitive recursion**). *Suppose the functions g_1, g_2, h_1 and h_2 are primitive recursive and define f_1 and f_2 by the system of equations*

$$f_1(0, \vec{x}) = g_1(\vec{x})$$
$$f_1(y+1, \vec{x}) = h_1(f_1(y, \vec{x}), f_2(y, \vec{x}), y, \vec{x})$$

$$f_2(0, \vec{x}) = g_2(\vec{x})$$
$$f_2(y+1, \vec{x}) = h_2(f_1(y, \vec{x}), f_2(y, \vec{x}), y, \vec{x}).$$

It follows that f_1 and f_2 are also primitive recursive.

PROOF in Problem x1B.17. ⊣

1B.16. PROPOSITION (**Complete primitive recursion**). *Suppose the function h is primitive recursive and let*

$$f(y, \vec{x}) = h(\langle f(0, \vec{x}), \dots, f(y \dot{-} 1, \vec{x})\rangle, y, \vec{x}),$$

so that $f(0, \vec{x}) = h(\langle\Lambda\rangle, 0, \vec{x})$, $f(1, \vec{x}) = h(\langle f(0, \vec{x})\rangle, 1, \vec{x})$, etc.. *It follows that f is also primitive recursive.*

PROOF. First we define by primitive recursion the function

$$g(0, \vec{x}) = \langle\Lambda\rangle,$$
$$g(y+1, \vec{x}) = g(y, \vec{x}) * \langle h(g(y, \vec{x}), y, \vec{x})\rangle,$$

and then we verify that the function

$$f(y, \vec{x}) = (g(y+1, \vec{x}))_y$$

satisfies the required equation. We finally show by complete induction on y that only one function satisfies the given system of equations. ⊣

Problems for Section 1B

1B.17. A **primitive recursive derivation** (or **program**) is any sequence of functions of several variables

$$E = (f_0, f_1, \dots, f_n)$$

such that for every $j \leq n$ one of the following is true:

(1) f_j is one of the basic primitive recursive functions S, P_i^n, C_q^n.

(2) f_j is defined by composition (10), where $g, h_1, \dots, h_m$ are basic primitive recursive functions or in the sequence $f_0, \dots, f_{j-1}$, before the j'th place in E.

(3) f_j is defined by primitive recursion (11), where g and h are basic primitive recursive functions or in the sequence $f_0, \dots, f_{j-1}$, before the j'th place in E.

x1B.1. Prove that $f : \mathbb{N}^k \to \mathbb{N}$ is primitive recursive if and only if $f = f_n$ for some primitive recursive derivation $(f_0, f_1, \dots, f_n)$.

x1B.2. Prove (directly, from the definitions) that if

$$f(x, y) = h(g_1(y), g_2(y, x), y)$$

and h, g_1, g_2 are primitive recursive, then f is also primitive recursive.

x1B.3. Prove that if $g : \mathbb{N}^2 \to \mathbb{N}$ is primitive recursive, then the function

$$f(x, y) = g(y, x)$$

is also primitive recursive.

x1B.4. Prove that for every $n \geq 1$, the n-place functions

$$\min_n(x_1, \dots, x_n) = \text{the least of } x_1, \dots, x_n$$
$$\max_n(x_1, \dots, x_n) = \text{the greatest of } x_1, \dots, x_n$$

are primitive recursive.

x1B.5. Prove that the exponential function $f(x, y) = x^y$ (with $0^0 = 1$) is primitive recursive.

x1B.6. Prove that the binary relations $x \leq y$, $x < y$ are primitive recursive.

x1B.7. Prove that the functions $\text{quot}(m, n)$ and $\text{rem}(m, n)$ are primitive recursive.

x1B.8. Prove (4) of Proposition 1B.6.

x1B.9. Prove (5) of Proposition 1B.6.

x1B.10. Prove Corollary 1B.7.

x1B.11. Prove that for every n, there exists a prime number p such that $n < p \leq n! + 1$. (One of the corollaries is that there exist infinitely many prime numbers, the so-called *Euclid's Theorem*.) Hint: If $n! + 1$ isn't prime, then some prime number $p \mid (n! + 1)$.

x1B.12. Prove that if the binary relation $R(x, y)$ and the function $g(x)$ are primitive recursive, then the function

$$f(x) = (\mu y \leq g(x))R(x, y)$$

is also primitive recursive.

The *greatest common divisor* of two numbers $x, y \geq 1$ is, well, the greatest number which divides both of them, when one common divisor exists:

$$\gcd(x,y) = \begin{cases} 0, & \text{if } x = 0 \text{ or } y = 0, \\ \text{the greatest } m \text{ such that} & \\ \quad m \mid x \text{ and } m \mid y, & \text{otherwise.} \end{cases} \tag{27}$$

x1B.13. Prove that the function $\gcd(x,y)$ is primitive recursive.

x1B.14. Why can we not define the notion of "primitive recursive coding of $\mathbb{N}^*$" with the simple

$$\text{1-1, primitive recursive function } \langle\ \rangle : \mathbb{N}^* \rightarrowtail \mathbb{N}$$

instead of the complicated (and several) conditions on which we based the definition?

x1B.15. Prove that the "classical coding" of $\mathbb{N}^*$ in Proposition 1B.13 is primitive recursive.

x1B.16. Prove that the following two functions (*restriction* and *concatenation*) are primitive recursive:

$$u\restriction i = \begin{cases} \langle u_0, \dots, u_{i-1}\rangle & \text{if } u = \langle u_0, \dots, u_{n-1}\rangle \text{ with } i \leq n, \\ 0, & \text{otherwise,} \end{cases}$$

$$u * v = \begin{cases} \langle u_0, \dots, u_{n-1}, v_0, \dots, v_{m-1}\rangle, & \text{if } u = \langle u_0, \dots, u_{n-1}\rangle, \\ & \quad v = \langle v_0, \dots, v_{m-1}\rangle, \\ 0, & \text{otherwise.} \end{cases}$$

x1B.17. Prove Proposition 1B.15.

x1B.18 (**Complete primitive recursion for relations**). Prove that if the relation $H(w, \vec{x})$ is primitive recursive and $P(y, \vec{x})$ satisfies the equivalence

$$P(y,\vec{x}) \iff H(\langle \chi_P(0,\vec{x}), \dots, \chi_P(y \dot{-} 1, \vec{x})\rangle, y, \vec{x}),$$

then $P(y, \vec{x})$ is also primitive recursive.

x1B.19* (**Nested recursion**). Prove that for any three functions $g(x)$, $h(w,x,y)$ and $\tau(x,y)$, there exists exactly one function $f(x,y)$ which satisfies the equations

$$f(0,y) = g(y), \qquad f(x+1,y) = h(f(x,\tau(x,y)),x,y);$$

and if the given functions are primitive recursive, then $f(x,y)$ is also primitive recursive.

x1B.20. Define a primitive recursive injection $g : \mathbb{N} \times \mathbb{N} \rightarrowtail \mathbb{N}$, such that

$$g(x, y) \leq (x + y + 1)^2.$$

More generally, show that for every $n \geq 2$, there exists a primitive recursive injection $g_n : \mathbb{N}^n \rightarrowtail \mathbb{N}$, such that

$$g_n(x_1, \dots, x_n) \leq P_n(x_1, \dots, x_n), \tag{28}$$

where $P_n(x_1, \dots, x_n)$ is a polynomial of degree n.

x1B.21. Prove that for every $n \geq 2$, there is no one-to-one function $g : \mathbb{N}^n \to \mathbb{N}$ which satisfies (28) with a polynomial of degree $\leq n - 1$.

x1B.22. Prove that there is a primitive recursive coding of sequences, such that for every n, and all $x_1, \dots, x_n$,

$$\langle x_1, \dots, x_n \rangle \leq 2^n P_n(x_1, \dots, x_n),$$

where the polynomial P_n is of degree n.

x1B.23. Prove that for every coding $\langle\ \rangle : \mathbb{N}^* \rightarrowtail \mathbb{N}$ of the sequences from $\mathbb{N}$,

$$\max\{\langle x_1, \dots, x_n \rangle \mid x_1, \dots, x_n \leq k\} \geq 2^n \quad (k, n \geq 2)$$

x1B.24*. (1) Prove that every section $A_n(x)$ of the Ackermann function is primitive recursive.

(2) Prove that for every primitive recursive function $f(x_1, \dots, x_n)$, there exists some m such that

$$f(x_1, \dots, x_n) < A_m(\max(x_1, \dots, x_n)) \quad (x_1, \dots, x_n \in \mathbb{N}). \tag{29}$$

(3) Prove that the Ackermann function $A(n, x)$ is not primitive recursive. HINT: Call a function $f(\vec{x})$ *A-bounded* if it satisfies (29) with some m, and prove that the collection of all A-bounded functions is primitively closed. Problems x1A.11 – x1A.14 provide the necessary Lemmas.

x1B.25. Prove that if the functions

$$\langle\ \rangle_1, \langle\ \rangle_2 : \mathbb{N}^* \rightarrowtail \mathbb{N}$$

are primitive recursive codings, then there exists a primitive recursive function $\pi : \mathbb{N} \to \mathbb{N}$, such that

$$\pi(\langle \vec{x} \rangle_1) = \langle \vec{x} \rangle_2 \quad (\vec{x} \in \mathbb{N}^*).$$

1C. μ-recursive partial functions

A number x is a *twin prime* if it is prime and $x + 2$ is also prime; for example, 5 is a twin prime but 7 is not. There exist exactly thirty five twin primes smaller than 1000, the following, each coupled with the next prime which follows it:

3:5	5:7	11:13	17:19	29:31
41:43	59:61	71:73	101:103	107:109
137:139	149:151	179:181	191:193	197:199
227:229	239:241	269:271	281:283	311:313
347:349	419:421	431:433	461:463	521:523
569:571	599:601	617:619	641:643	659:661
809:811	821:823	827:829	857:859	881:883

The conjecture that *there exist infinitely many twin primes* is a famous open problem, and (as number theorists tell us) there is no realistic expectation that it will be proved soon. Suppose that it is true and define the function

$$p_i^t = \text{the } i\text{-th twin prime number},$$

so that (by the table),

$$p_0^t = 3,\ p_9^t = 107,\ p_{34}^t = 881.$$

The function p_i^t obviously satisfies the recursive equation

$$\begin{aligned} p_0^t &= 3, \\ p_{i+1}^t &= h(p_i^t + 1), \end{aligned}$$

where

$$h(w) = \text{the least twin prime } x \geq w = (\mu y \geq w)\text{Prime}^t(y), \tag{30}$$

and the relation

$$\text{Prime}^t(x) \iff x \text{ is a twin prime}$$

is primitive recursive. This however does not imply that the function p_i^t is primitive recursive (as in the proof of the analogous Proposition 1B.10 for the function p_i), because we cannot show that the function $h(w)$ is primitive recursive—and this because we don't know some bound for the next twin prime. On the other hand, the function $h(w)$ can be computed by an obvious *dumb search*, where we successively check the conditions

$$\text{Prime}^t(w+1),\ \text{Prime}^t(w+2),\ \text{Prime}^t(w+3), \dots,$$

until we find some $w + 1 + i$ which is, indeed, a twin prime. For example,

$$\begin{aligned} h(6) &= h(7) \quad \text{because } \neg\text{Prime}^t(6) \\ &= h(8) \quad \text{because } \neg\text{Prime}^t(7) \\ &= h(9) \quad \text{because } \neg\text{Prime}^t(8) \\ &= h(10) \quad \text{because } \neg\text{Prime}^t(9) \\ &= h(11) \quad \text{because } \neg\text{Prime}^t(10) \\ &= 11 \quad \text{because Prime}^t(11). \end{aligned}$$

and then the function p_i^t is computed by the primitive recursion which defines it from h,[2]

$$p_0^t = 3,\ p_1^t = h(p_0^t + 1),\ \ldots\ , p_i^t = h(p_{i-1}^t + 1).$$

Despite its derogatory name, dumb search is perhaps the most basic procedure in the construction of algorithms on natural numbers. It is expressed by the (**unbounded**) **minimalization operator** which is applied to a relation like bounded minimalization in 1B.8:

$$(31)\quad (\mu i \geq y)R(i,\vec{x}) \\ = \text{the least } i \geq y \text{ (if there exists one) such that } R(i,\vec{x}),$$

and somewhat more simply for a dumb search which starts from 0,

$$\mu i R(i,\vec{x}) = (\mu i \geq 0)R(i,\vec{x}).$$

Its application, however, leads naturally to the introduction of "partial functions" which do not always deliver a value, and we give its precise definition in this wider context.

1C.1. DEFINITION. A **partial function** (from the *input set* A to the *output set* or *range* B)

$$f : A \rightharpoonup B \quad \text{(note the half-arrow)}$$

is any function

$$f : A_0 \to B \quad (A_0 = \text{Domain}(f) \subseteq A)$$

from a subset A_0 of A, its *domain of convergence*. We write

$$f(x)\!\downarrow \iff x \in \text{Domain}(f) \quad (f(x)\ \textit{converges})$$
$$f(x)\!\uparrow \iff x \notin \text{Domain}(f) \quad (f(x)\ \textit{diverges}),$$

and occasionally, in definitions, the ungrammatical

$$f(x) = \uparrow \ \text{ which means that } f(x)\!\uparrow .$$

For n-place partial functions $f, g : \mathbb{N}^n \rightharpoonup \mathbb{N}$ on the natural numbers, we also write

$$f(\vec{x}) > 0 \iff f(\vec{x})\!\downarrow \ \ \&\ f(\vec{x}) > 0,$$
$$f(\vec{x}) < g(\vec{y}) \iff f(\vec{x})\!\downarrow \ \ \&\ g(\vec{y})\!\downarrow \ \ \&\ f(\vec{x}) < g(\vec{y}),$$

etc. The extreme case of a partial function is the totally undefined (empty) function with

$$(32)\qquad \text{Domain}(\varepsilon) = \emptyset, \text{ so that for all } x, \varepsilon(x)\!\uparrow .$$

[2]We should note that the program which created the table of twin primes at the beginning of this section is based on a much more efficient algorithm which uses the "sieve of Eratosthenes".

On the other hand, every (ordinary, *total*) function $f : A \to B$ is a partial function, with Domain$(f) = A$.

If $f, g : A \rightharpoonup B$ and $x \in A$, we write

$$f(x) = g(x) \iff \Big(f(x)\uparrow \ \& \ g(x)\uparrow\Big) \vee f(x) = g(x),$$

so that with the ungrammatical use of $f(x) = \uparrow$ above,

$$f = g \iff (\forall x)[f(x) = g(x)].$$

We also set

$$f \sqsubseteq g \iff (\forall \vec{x})[f(\vec{x})\downarrow \implies f(\vec{x}) = g(\vec{x})]. \tag{33}$$

The relation $f \sqsubseteq g$ is (easily, x1C.2) a *partial ordering* on the set

$$(A \rightharpoonup B) = \{f \mid f : A \rightharpoonup B\}$$

of all partial functions with input set A and range B, that is

$$f \sqsubseteq f, \quad \Big(f \sqsubseteq g \ \& \ g \sqsubseteq h\Big) \implies f \sqsubseteq h, \quad \Big(f \sqsubseteq g \ \& \ g \sqsubseteq f\Big) \implies f = g.$$

1C.2. **Composition and primitive recursion**. These operators are interpreted for partial functions by the natural way that we compute them: if, e.g., $g, h : A \rightharpoonup B$ and $f : B^2 \rightharpoonup C$, then for all $x \in A, w \in C$,

$$f(g(x), h(x)) = w \iff (\exists u, v \in B)[g(x) = u \ \& \ h(x) = v \ \& \ f(u, v) = w];$$

and if

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

and g, h are partial functions on $\mathbb{N}$, then, for all $y, \vec{x}, w \in \mathbb{N}$,

$$\begin{aligned} f(y, \vec{x}) = w \iff (\exists w_0, \dots, w_y \in \mathbb{N}) & \\ & \Big(w_0 = g(\vec{x}) \\ & \ \& \ (\forall i, 0 < i \leq y)[w_i = h(w_{i \dot{-} 1}, i \dot{-} 1, \vec{x})] \\ & \ \& \ w_y = w\Big). \end{aligned} \tag{34}$$

It follows from this definition that

$$g(\vec{x})\uparrow \implies (\forall y)[f(y, \vec{x})\uparrow],$$

because, for each y, the computation of the value $f(y, \vec{x})$ ultimately depends on the computation of $f(0, \vec{x}) = g(\vec{x})$.

Equivalence (34) gives an *explicit definition* for the partial function $f(y, \vec{x})$ and is known as *Dedekind's analysis of recursion*. Note that it uses the quantifier $(\exists w_0, \dots, w_y \in \mathbb{N})$ on *finite sequences of natural numbers*.

With these notions, we can extend the definition of the set $\mathcal{R}_p(\Psi)$ of Ψ-*primitive recursive functions* to the case where Ψ contains partial functions. We use this in 1C.4 below.

1C.3. DEFINITION. The **minimalization** of the partial function $g(i,\vec{x})$ is the partial function

$$\begin{aligned}(35)\quad f(y,\vec{x}) &= (\mu i \geq y)[g(i,\vec{x}) = 0] \\ &= \text{the least } i \geq y, \text{ such that} \\ &\qquad g(i,\vec{x}) = 0 \ \&\ (\forall j \geq y)[j < i \Longrightarrow g(j,\vec{x})\downarrow \ \&\ g(j,\vec{x} \neq 0].\end{aligned}$$

Equivalently:

$$\begin{aligned}&(\mu i \geq y)[g(i,\vec{x}) = 0] = w \\ &\qquad \Longleftrightarrow\ g(w,\vec{x}) = 0 \ \&\ (\forall j \geq y)[j < w \Longrightarrow g(j,\vec{x})\downarrow \ \&\ g(j,\vec{x}) \neq 0].\end{aligned}$$

For example,

$$g(1)\uparrow \ \Longrightarrow (\mu i \geq 1)[g(i) = 0]\uparrow,$$

even if $g(2) = 0$. We also observe that if $g(i,\vec{x})$ is a total function and we set

$$R(i,\vec{x}) \ \Longleftrightarrow\ g(i,\vec{x}) = 0,$$

then $(\mu i \geq y)[g(i,\vec{x}) = 0] = (\mu i \geq y)R(i,\vec{x})$ according to definition (31).

1C.4. DEFINITION. A partial function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ is **μ-recursive** in the set of partial functions Ψ if f belongs to every set of partial functions which contains Ψ and is primitively closed and closed under minimalization. In symbols:

$$\mathcal{R}_\mu(\Psi) = \{f \mid f \text{ is } \mu\text{-recursive in } \Psi\}, \quad \mathcal{R}_\mu = \mathcal{R}_\mu(\emptyset).$$

A relation $R(\vec{x})$ or set $A \subseteq \mathbb{N}$ is μ-recursive if its characteristic function is μ-recursive.

The set $\mathcal{R}_\mu(\Psi)$ of partial functions which are μ-recursive in Ψ is closed under primitive recursion by its definition, i.e.,

$$\mathcal{R}_p(\Psi) \subseteq \mathcal{R}_\mu(\Psi).$$

We will show later that the converse inclusion does not hold, but this is not obvious now. We can prove immediately, however, the closure of $\mathcal{R}_\mu(\Psi)$ under the *branching operator*, which is especially useful when we apply it to partial functions:

1C.5. DEFINITION. The **branching** of three, given partial functions $c(\vec{x})$, $g(\vec{x})$, $h(\vec{x})$, is the partial function

(36) $f(\vec{x}) = \text{if } (c(\vec{x}) = 0) \text{ then } g(\vec{x}) \text{ else } h(\vec{x})$

$$= \begin{cases} g(\vec{x}), & \text{if } c(\vec{x}) = 0, \\ h(\vec{x}), & \text{if } c(\vec{x})\downarrow \ \& \ c(\vec{x}) \neq 0, \\ \uparrow, & \text{if } c(\vec{x})\uparrow, \end{cases}$$

with the convergence condition

$$f(\vec{x})\downarrow \iff [c(\vec{x}) = 0 \ \& \ g(\vec{x})\downarrow] \vee [c(\vec{x})\downarrow \ \& \ c(\vec{x}) > 0 \ \& \ h(\vec{x})\downarrow].$$

Note that the convergence of branching requires the convergence of the *test* $c(\vec{x})$ but does not require the convergence of both values $g(\vec{x})$ and $h(\vec{x})$: for example, whether $g(x)\downarrow$ or $g(x)\uparrow$,

$$\text{if } (0 = 0) \text{ then } x \text{ else } g(x) = x.$$

1C.6. PROPOSITION. *For every set of partial functions Ψ, the sets $\mathcal{R}_p(\Psi)$ and $\mathcal{R}_\mu(\Psi)$ are closed under branching.*

PROOF. For the given definition

$$f(\vec{x}) = \text{if } (c(\vec{x}) = 0) \text{ then } g(\vec{x}) \text{ else } h(\vec{x}),$$

the first temptation is to set, by primitive recursion,

$$\begin{aligned} f_1(0, \vec{x}) &= g(\vec{x}), \\ f_1(i+1, \vec{x}) &= h(\vec{x}), \end{aligned}$$

and try to show

$$f(\vec{x}) = f_1(c(\vec{x}), \vec{x}), \tag{37}$$

so that if $c, g, h \in \mathcal{R}_p(\Psi)$, then $f \in \mathcal{R}_p(\Psi) \subseteq \mathcal{R}_\mu(\Psi)$. This does not work, Problem x1C.5, and we set instead, successively,

$$\begin{aligned} \varphi_0(0, \vec{x}) &= 0, & \varphi_1(0, \vec{x}) &= 0, \\ \varphi_0(i+1, \vec{x}) &= g(\vec{x}), & \varphi_1(i+1, \vec{x}) &= h(\vec{x}), \\ f_1(\vec{x}) &= \varphi_0(1 \dot{-} c(\vec{x})) + \varphi_1(c(\vec{x}), \vec{x})) \end{aligned}$$

and now the equation (37) follows easily, Problem x1C.5. ⊣

1C.7. **Recursive equations**. The Ackermann function (1A.6) is not primitive recursive, and it is not obvious (yet) whether it is μ-recursive. Problem x1B.19* gives another interesting example of a function for which it is not easy to show that it is primitive recursive, or μ-recursive for that matter. These functions, however, are *computable*, because the recursive equations which determine them yield algorithms for the computation of their values, e.g., in Problems x1A.6 and x1A.8. The next Proposition gives one more (somewhat peculiar) example of this kind.

1C.8. PROPOSITION. *The minimalization*

$$f(y,\vec{x}) = (\mu i \geq y)[g(i,\vec{x}) = 0]$$

of a partial function g is the $\sqsubseteq$-least solution of the recursive equation

$$p(y,\vec{x}) = \text{if } (g(y,\vec{x}) = 0) \text{ then } y \text{ else } p(y+1,\vec{x}), \tag{38}$$

that is:

(1) *The recursive equation* (38) *holds for all $y,\vec{x}$ if we set $p := f$.*

(2) *If some partial function p satisfies* (38) *for all $y,\vec{x}$, then $f \sqsubseteq p$.*

PROOF. (1) We have to show that for all $y,\vec{x}$,

$$f(y,\vec{x}) = \text{if } (g(y,\vec{x}) = 0) \text{ then } y \text{ else } f(y+1,\vec{x}), \tag{39}$$

and we consider the three cases,

$$g(y,\vec{x})\uparrow,\ g(y,\vec{x}) = 0,\ g(y,\vec{x}) > 0.$$

The truth of (39) is obvious in the first two of them. In the third case, if

$$(\forall i \geq y)[g(i,\vec{x})\uparrow \vee g(i,\vec{x}) > 0],$$

then, obviously,

$$f(y,\vec{x})\uparrow \text{ and } f(y+1,\vec{x})\uparrow$$

so that we have equality again. Finally, if, for some $w > y$,

$$g(w,\vec{x}) = 0 \ \&\ (\forall i \geq y)[i < w \Longrightarrow g(i,\vec{x}) > 0],$$

then $f(y,\vec{x}) = f(y+1,\vec{x}) = w$, and we have again equality.

(2) We need to show that if for all $\vec{x},y$,

$$p(y,\vec{x}) = \text{if } (g(y,\vec{x}) = 0) \text{ then } y \text{ else } p(y+1,\vec{x}), \tag{40}$$

then, for every $\vec{x}$ and all y and $w \geq y$,

$$f(y,\vec{x}) = w \Longrightarrow p(y,\vec{x}) = w.$$

We use induction on the difference $w - y$.

At the BASIS, $w = y$, and immediately, by the definition of f and (40),

$$f(y,\vec{x}) = y = p(y,\vec{x}).$$

At the INDUCTION STEP, $w > y$ and $g(y,\vec{x}) > 0$, so that

$$\begin{aligned} w = f(y,\vec{x}) &= f(y+1,\vec{x}) && \text{(by (39))} \\ &= p(y+1,\vec{x}) && \text{(ind. hyp.,} \\ & && \text{since } w-(y+1) = (w-y)-1), \\ &= p(y,\vec{x}) && \text{(by (40)).} \end{aligned}$$

⊣

This example is especially important because it shows that the equation (38) which seems to define the value $p(y, \vec{x})$ from the next $p(y+1, \vec{x})$ (!) in fact expresses the basic dumb search procedure. For example, to compute the value $p(2, \vec{x})$ by this equation, we apply it repeatedly,

$$p(2, \vec{x}) = p(3, \vec{x}) = \cdots$$

until (perhaps) we find some m such that $g(m, \vec{x}) = 0$, in which case we know that $p(2, \vec{x}) = m$. The example shows that the relation between *recursion* and *computation* which we pointed out in (II) of 1A.3 applies much more widely than the classical case of primitive recursion. It is the key idea for defining the fundamental class of (general) *recursive partial functions* which we will take up in the next Chapter.

Problems for Section 1C

x1C.1. Prove that for all partial functions $f, g : A \rightharpoonup B$,

$$f = g \Longrightarrow (\forall x \in A)[f(x)\downarrow \iff g(x)\downarrow] \tag{1}$$

and

$$f = g \iff (\forall x \in A, w \in B)[f(x) = w \iff g(x) = w]. \tag{2}$$

x1C.2. Prove that for all partial functions $f, g, h : A \rightharpoonup B$,

$$f \sqsubseteq f, \quad [f \sqsubseteq g \ \& \ g \sqsubseteq h] \Longrightarrow f \sqsubseteq h, \quad [f \sqsubseteq g \ \& \ g \sqsubseteq f] \Longrightarrow f = g.$$

x1C.3. Consider the definitions

$$\begin{aligned} g(x, y, z) &= \text{if } (x = 0) \text{ then } y \text{ else } z, \\ f_1(t) &= g(t, h(t), t), \\ f_2(t) &= \text{if } (t = 0) \text{ then } h(t) \text{ else } t \end{aligned}$$

where $h(t)$ is some partial function. Is the equation

$$f_1(t) = f_2(t)$$

true for every t? (Give a proof or a counterexample.)

x1C.4. Prove that if the relation $R(i, \vec{x})$ is μ-recursive, then the partial function

$$f(y, \vec{x}) = (\mu i \geq y) R(i, \vec{x})$$

is also μ-recursive.

x1C.5. Explain why the first idea for the proof of 1C.6 does not work, and give the details of the correct proof.

x1C.6. (1) Prove that the partial function $\varepsilon^n : \mathbb{N}^n \rightharpoonup \mathbb{N}$ of n variables with $\text{Domain}(\varepsilon^n) = \emptyset$ is μ-recursive.

(2) Suppose $g(\vec{x}), h(\vec{x})$ are μ-recursive partial functions and let

$$f(\vec{x}) = \begin{cases} g(\vec{x}) & \text{if } h(\vec{x})\downarrow, \\ \uparrow & \text{otherwise.} \end{cases}$$

This means that

$$\text{Domain}(f) = \text{Domain}(g) \cap \text{Domain}(h),$$

and for $x \in \text{Domain}(f)$, $f(x) = g(x)$. Prove that f is also μ-recursive.

x1C.7. Prove that the classes $\mathcal{R}_p(\Psi)$ and $\mathcal{R}_\mu(\Psi)$ are closed under definitions with $k+1$ cases, of the form

$$f(\vec{x} = \begin{cases} g_1(\vec{x}), & \text{if } c_1(\vec{x}) = 0, \\ g_2(\vec{x}), & \text{if } c_1(\vec{x}) > 0 \ \&\ c_2(\vec{x}) = 0, \\ \cdots & \\ g_k(\vec{x}), & \text{if } c_1(\vec{x}) > 0, \dots, c_{k-1}(\vec{x}) > 0, c_k(\vec{x}) = 0, \\ g_{k+1}(\vec{x}) & \text{if } c_1(\vec{x}) > 0, \dots, c_{k-1}(\vec{x}) > 0, c_k(\vec{x}) > 0. \end{cases}$$

x1C.8*. (1) Prove that there exists exactly one partial function $p(x, y)$ which satisfies the equations

$$\begin{aligned} p(0, y) &= y, \\ p(x+1, 0) &= 2x+1, \\ p(x+1, y+1) &= 3p(x, y) + p(y, x) + 2, \end{aligned}$$

and compute the value $p(3, 2)$ for this p.

(2) Prove that the unique solution of this system is primitive recursive (or μ-recursive, if this is easier).

x1C.9. For partial functions $g(\vec{x})$, $h(w, y, \vec{x})$, prove that the recursive equation

$$p(y, \vec{x}) = \text{if } (y = 0) \text{ then } g(\vec{x}) \text{ else } h(p(y \dot{-} 1, \vec{x}), y \dot{-} 1, \vec{x}) \tag{41}$$

has exactly one solution, namely the partial function f which is defined by the primitive recursion

$$\begin{aligned} f(0, \vec{x}) &= g(\vec{x}), \\ f(y+1, \vec{x}) &= h(f(y, \vec{x}), y, \vec{x})). \end{aligned}$$

x1C.10 (The Euclidean algorithm). Prove that the following recursive equation has a unique solution, the function $\overline{p}(x,y) = \gcd(x,y)$:

$$p(x,y) = \begin{cases} 0, & \text{if } x = 0 \text{ or } y = 0, \\ p(y,x), & \text{otherwise, if } x < y, \\ y, & \text{otherwise, if } y \mid x, \\ p(y, \operatorname{rem}(x,y)), & \text{otherwise.} \end{cases} \tag{42}$$

x1C.11. Give examples of partial functions $g(y,\vec{x})$ such that equation (38) admits:

(1) The empty partial function ε as the only solution.
(2) Only one solution, which is a total function.
(3) Infinitely many solutions.

x1C.12*. Prove that there is no partial function $g(y,\vec{x})$ such that equation (38) has exactly two solutions.

x1C.13*. (1) Prove that the recursive equation

$$p(x,y) = \text{if } (x = 0) \text{ then } 1 \text{ else } p(x \dot{-} 1, p(x,y)) \tag{$*$}$$

has a least solution $\overline{p}$, and give a formula for this solution.

(2) Prove that there exists only one total function f which satisfies the recursive equation $(*)$, and this function is not the least solution.

CHAPTER 2

GENERAL RECURSION

The first rigorous definition of (general) **recursive partial functions** on the natural numbers was given by Gödel in 1934, following a suggestion of Herbrand, and the basic results about recursive functions were proved by Kleene in the 1930's. We will develop a generalization to *partial algebras* of a very simple and elegant version of the Gödel-Herbrand-Kleene definition. It is due to John McCarthy (1963) and it reveals more clearly the relationship between recursion and computation.

This approach to recursion theory is based on ideas from logic and programming, and in the first section of the chapter we will prepare the ground by presenting (briefly) the required preliminaries.

2A. Partial algebras

Characteristic—and for us the main—example of a partial algebra is the *basic structure of arithmetic*

$$\mathbf{N}_0 = (\mathbb{N}, 0, 1, S, Pd), \tag{43}$$

where S and Pd are the successor and the predecessor functions on the natural numbers. In general:

2A.1. DEFINITION. A (*partial*) **algebra** is a tuple

$$\mathbf{M} = (M, 0, 1, f_1, \dots, f_K),$$

where M is a set, the *universe* of $\mathbf{M}$; $0, 1 \in M$ and $0 \neq 1$; and for each $i = 1, \dots, K$,

$$f_i : M^{n_i} \rightharpoonup M$$

is a partial function of arity n_i. Here we allow $n_i = 0$, in which case f_i is a 0-place partial function on M, i.e., a *constant* (some member of M) or the totally undefined partial function of 0 arguments.

An algebra is **total** if every f_i is a total function.

Expansions. In almost everything we do, we will only need algebras on the natural numbers, including *expansions* of $\mathbf{N}_0$ of the form

$$(\mathbf{N}_0, f_1, \dots, f_m) = (\mathbb{N}, 0, 1, S, Pd, f_1, \dots, f_m)$$

and the standard *first order structure on* $\mathbb{N}$

$$\mathbf{N} = (\mathbb{N}, 0, 1, +, \cdot). \tag{44}$$

Other basic examples of partial algebras are the structures

$$(\mathbb{Z}, 0, 1, +, -, \cdot), \qquad (F, 0, 1, +, -, \cdot, \div),$$

where $\mathbb{Z} = \{0, 1, -1, 2, -2, \dots\}$ is the set of (positive and negative) integers, and F is a *field*, e.g.,

$$F = \mathbb{Q} = \text{ the set of rational numbers } = \left\{ \frac{x}{y} \mid x, y \in \mathbb{Z}, y \neq 0 \right\}$$
$$\text{or } F = \mathbb{R} = \text{ the set of real numbers.}$$

Division $r \div s$ is a partial function on a field (since it does not converge when $s = 0$), and so fields are not total algebras.[3]

2A.2. **The term language $\mathsf{T}(\mathbf{M})$: syntax**. With every partial algebra $\mathbf{M}$ we associate the formal language $\mathsf{T} = \mathsf{T}(\mathbf{M})$ of terms, whose **alphabet** comprises the following **symbols**:

- the *individual variables*: $\mathsf{v}_0, \mathsf{v}_1, \dots,$
- the *individual constants* (members of M): x $(x \in M)$
- the *function constants*: $\mathsf{f}_1, \dots, \mathsf{f}_K$ $(\text{arity}(\mathsf{f}_i) = n_i)$
- the *symbols for branching*: $\mathsf{if} \quad \mathsf{then} \quad \mathsf{else}$
- the *punctuation symbols*: $, \quad (\quad)$
- and the symbol for equality: $=$

From these symbols we single out the **vocabulary** of $\mathbf{M}$

$$\boldsymbol{v} = (\mathsf{f}_1, \dots, \mathsf{f}_K) \qquad (\text{arity}(\mathsf{f}_i) = n_i) \tag{45}$$

[3] In logic we study structures

$$\mathbf{M} = (M, c_1, \dots, c_N, R_1, \dots, R_L, f_1, \dots, f_K),$$

where the *primitives* are distinguished elements of M, relations, and (total) functions on M. The *partial algebras* that we are using here are seemingly more special, as we do not allow distinguished elements or relations; but we can represent members of M by 0-place functions and n-ary relations by their characteristic functions. So partial algebras are in fact more general than the structures of first order logic, as we allow *partial* (not necessarily total) functions $f_1, \dots, f_K$. Those who know logic should also notice that in the definition of *terms* below, we allow *branching*.

which provides notation for the "given" (primitive) partial functions of $\mathbf{M}$, while the remaining symbols are common to all partial algebras on M. The vocabulary of $\mathbf{N}_0$ is the pair $(\mathsf{S}, \mathsf{Pd})$.

Words from a set Σ are finite sequences of elements (or *symbols*) of Σ, for example the words

$$\mathsf{v}_{17} \text{ (of length 1)} \qquad 0{=}(0\mathsf{v}_3\mathsf{f}_2 \qquad \mathsf{Pd}01x_1x_3\mathsf{if}{=}\mathsf{else}$$

from the alphabet of T. As in these examples, we will write words by simply arranging the symbols in a row, without commas. The *concatenation* of two words is denoted (literally) by the concatenation of their symbols, so that for the two words we gave as examples, their concatenation is the word

$$0{=}(0\mathsf{v}_3\mathsf{f}_2\mathsf{Pd}01x_1x_3\mathsf{if}{=}\mathsf{else}$$

We use the symbol '$\equiv$' to denote the relation of **word equality**, so that

$$\mathsf{f}_2{=}(0\mathsf{f}_1 \equiv \mathsf{f}_2{=}(0\mathsf{f}_1 \text{ but } \mathsf{f}_2{=}(0\mathsf{f}_2 \not\equiv \mathsf{f}_2{=}(1\mathsf{f}_2$$

The obvious reason for using "$\equiv$" for word equality is that '$=$' is a symbol (of the alphabet of T), so its use for the equality between words would cause confusion.

We often use Greek letters to name symbols and words,

$$\alpha \equiv \alpha_0\alpha_1 \cdots \alpha_m$$

The **empty word** is denoted by 'Λ', so that for every word α,

$$\Lambda\alpha \equiv \alpha\Lambda \equiv \alpha$$

The "syntactically correct" expressions of T—those which mean something—are separated into **terms** and **equations**. We define first the terms, following the same idea that we used to define primitive recursive functions.

2A.3. Definition (Terms and subterms). A set T of words from the alphabet of $\mathsf{T}(\mathbf{M})$ is **closed under term formation** if it satisfies the following conditions:

(T1) T contains every $x \in M$, including 0 and 1.
(T2) T contains every individual variable v_i.
(T3) If the words $A_1, \ldots, A_{n_i}$ are in T, then the word

$$\mathsf{f}_i(A_1, \ldots, A_{n_i})$$

is also in T.
(T4) If the words A_1, A_2, A_3 are in T, then the word

$$(\mathsf{if}\ (A_1 = 0)\ \mathsf{then}\ A_2\ \mathsf{else}\ A_3)$$

is also in T.

A word E is a **term** if it belongs to every set T which is closed under term formation, and we let

$$\text{TERMS} = \text{TERMS}(\mathbf{M}) = \text{ the set of all terms of } \mathsf{T}(\mathbf{M}).$$

A word A is a **subterm** of a term E if it is a term and a part of E, i.e.,

$$A \equiv \sigma A \tau$$

for suitable (possibly empty) words σ, τ. In particular, every term E is a subterm of itself.

Notice that the empty word Λ is not a term: because the set T of all non-empty words is (trivially) closed under term formation.

The definition of terms is often expressed informally by the "equivalence"

$$A :\equiv x \mid \mathsf{v}_i \mid \mathsf{f}_i(A_1, \dots, A_{n_i}) \mid (\mathsf{if}\ (A_1 = 0)\ \mathsf{then}\ A_2\ \mathsf{else}\ A_3) \tag{46}$$

which we read as follows:

> A word A is a term if it is a member of M, or an individual variable, or of the form $\mathsf{f}_i(A_1, \dots, A_{n_i})$ where $A_1, \dots, A_{n_i}$ are terms, or of the form $(\mathsf{if}\ (A_1 = 0)\ \mathsf{then}\ A_2\ \mathsf{else}\ A_3)$ where A_1, A_2, A_3 are terms—*and only if A satisfies one of these conditions.*

2A.4. **Pure and closed terms**. A term A is **closed** if no individual variable occurs in A, and **pure** if the only individual constants which (possibly) occur in A are 0 and 1. Examples in $\mathsf{T}(\mathbf{N}_0)$:

$$\mathsf{S}(0) : \text{closed, pure}, \quad \mathsf{S}(13) : \text{closed, not pure},$$
$$\mathsf{Pd}(\mathsf{v}_1) : \text{pure, not closed},$$
$$(\mathsf{if}\ (\mathsf{v}_1 = 0)\ \mathsf{then}\ 3\ \mathsf{else}\ 1) : \text{neither closed nor pure}$$

The recursive definition of terms justifies inductive proofs of properties of them:

2A.5. Lemma (Induction on terms). *Let Σ be a set of words in the alphabet of $\mathsf{T}(\mathbf{M})$ such that*

$$A_1, \dots, A_{n_i} \in \Sigma \Longrightarrow \mathsf{f}_i(A_1, \dots, A_{n_i}) \in \Sigma,$$
$$A, B, C \in \Sigma \Longrightarrow (\mathsf{if}\ (A = 0)\ \mathsf{then}\ B\ \mathsf{else}\ C) \in \Sigma.$$

(1) *If Σ contains all the individual constants and variables, then Σ contains all the terms.*

(2) *If Σ contains all the individual constants, then Σ contains all the closed terms.*

(3) *If Σ contains 0, 1 and all the individual variables, then Σ contains all the pure terms.*

PROOF. (1) is immediate: because the hypothesis says that Σ is closed under term formation, and so it includes every term, by the definition.

We leave (2) and (3) for Problem x2A.2. ⊣

How can we show that the word $(\mathsf{S}(1)$ is not a term? The rigorous proof is based on part (1) of the next technical but important Lemma, where the word α is a *proper initial segment* of the word β if

$$\beta \equiv \alpha\gamma \text{ with } \beta \not\equiv \Lambda,\ \gamma \not\equiv \Lambda$$

2A.6. LEMMA (Unique readability of terms). (1) *For every term A, the number of occurrences of the left parenthesis '(' in A is equal to the number of occurrences of the right parenthesis ')' in A.*

(2) *No proper initial segment of a term is a term.*

(3) *The set of terms is closed under term formation, and if $A \equiv \alpha_1 \cdots \alpha_m$ is a term, then exactly one of the cases* (T1) – (T4) *applies with T the set of terms.*

PROOF in Problem x2A.3. ⊣

The word $(\mathsf{S}(1)$ is obviously not a term, since it has more left parentheses than right ones.

Lemma 2A.6 also justifies *recursive definitions* of functions on the terms.

2A.7. LEMMA (Recursion on terms). *For each set W and given functions Φ_1, Φ_2^i $(i = 1, \ldots, K)$, Φ_3, there exists a unique function*

$$\Phi : \text{TERMS} \to W$$

such that

$$\begin{aligned} \text{for } x \in M,\ \Phi(x) &= \Phi_1(x), \quad \Phi(\mathsf{v}_i) = \Phi_1(\mathsf{v}_i), \\ \Phi(\mathsf{f}_i(A_1, \ldots, A_{n_i})) &= \Phi_2^i(\Phi(A_1), \ldots, \Phi(A_{n_i})), \\ \Phi(\mathsf{if}\ (A = 0)\ \mathsf{then}\ B\ \mathsf{else}\ C) &= \Phi_3(\Phi(A), \Phi(B), \Phi(C)). \end{aligned}$$

PROOF in Problem x2A.5. ⊣

Similar results hold for the sets of closed and of pure terms, justifying recursive definitions on them, Problem x2A.6.

2A.8. DEFINITION (Equations). An **equation** of $\mathbf{M}$ is a word of the form

$$A = B$$

where A and B are terms of $\mathbf{M}$.

Notice that the *syntax* of $\mathsf{T}(\mathbf{M})$ is completely determined by the set M and the vocabulary $\boldsymbol{v} = (\mathsf{f}_1, \ldots, \mathsf{f}_K)$ of $\mathbf{M}$, that is, it does not depend on the interpretations $f_1, \ldots, f_K$ of the function symbols in $\mathbf{M}$. These are used to define the *semantics* of $\mathsf{T}(\mathbf{M})$, as follows.

2A.9. **The term language $\mathsf{T}(\mathbf{M})$: semantics**. The **value** $\mathbf{val}_{\mathbf{M}}(A)$ of a **closed term** A in the partial algebra $\mathbf{M}$ is defined recursively by the following conditions by appealing to Lemma 2A.7 for closed terms, cf. Problem x2A.6:

(DT1) $\mathbf{val}_{\mathbf{M}}(x) = x$, for $x \in M$.
(DT2) If $\mathbf{val}_{\mathbf{M}}(A_j) = w_j$ for $j = 1, \dots, n_i$, then
$\mathbf{val}_{\mathbf{M}}(\mathsf{f}_i(A_1, \dots, A_{n_i})) = f_i(w_1, \dots, w_{n_i})$.
(DT3) If $E \equiv (\mathsf{if}\ (A = 0)\ \mathsf{then}\ B\ \mathsf{else}\ C)$, then

$$\mathbf{val}_{\mathbf{M}}(E) \equiv \begin{cases} \mathbf{val}_{\mathbf{M}}(B) & \text{if } \mathbf{val}_{\mathbf{M}}(A) = 0, \\ \mathbf{val}_{\mathbf{M}}(C) & \text{if } \mathbf{val}_{\mathbf{M}}(A)\downarrow \ \ \& \ \mathbf{val}_{\mathbf{M}}(A) \neq 0. \end{cases}$$

Notice that the definition can yield $\mathbf{val}_{\mathbf{M}}(E)\uparrow$, since the primitives of the algebra $\mathbf{M}$ are allowed to be partial functions: for example, if $f_1(1)\uparrow$, then $\mathbf{val}_{\mathbf{M}}(\mathsf{f}_1(1)) = f_1(1)\uparrow$.

2A.10. **Simplified notation** (misspellings). As is usual in logic, we rarely put down "grammatically correct" terms and equations. In practice, we use the ordinary mathematical symbols instead of their formal counterparts, e.g., $x, y, \dots$ or sometimes $\mathsf{x}, \mathsf{y}, \dots$ for individual variables instead of $\mathsf{v}_1, \mathsf{v}_2, \dots$, $f, g, +, \cdot, p, \dots$ for function constants instead of $\mathsf{f}_1, \mathsf{f}_2, \dots$, etc. We also omit or introduce more parentheses and "blank spaces" if this facilitates readability and we use *infix notation*, writing for example

$$(x + y) \cdot z \text{ instead of } \cdot(+(x, y), z)$$

The "grammatically correct" expression for (38) (with the suitable vocabulary) is

$$\begin{aligned} &\mathsf{p}(\mathsf{v}_1, \mathsf{v}_2, \dots, \mathsf{v}_{n+1}) \\ &\quad = (\mathsf{if}\ (\mathsf{f}_1(\mathsf{v}_1, \mathsf{v}_2, \dots, \mathsf{v}_{n+1}) = 0)\ \mathsf{then}\ \mathsf{v}_1\ \mathsf{else}\ \mathsf{p}(\mathsf{S}(\mathsf{v}_1), \mathsf{v}_2, \dots, \mathsf{v}_{n+1})) \end{aligned}$$

while its simplified rendering in $\mathsf{T}(\mathbf{N}_0, g, p)$ is

$$p(y, \vec{x}) = \text{if } (g(y, \vec{x}) = 0) \text{ then } y \text{ else } p(S(y), \vec{x}). \tag{47}$$

Definitely more readable.

2A.11. **Substitution of constants**. For each term A, each individual variable x and each $x \in M$, we set

$$A\{\mathsf{x} :\equiv x\} \equiv \text{the result of replacing } \mathsf{x} \text{ by } x \text{ in the term } A.$$

More generally, for any distinct individual variables $\vec{\mathsf{x}} = \mathsf{x}_1, \dots, \mathsf{x}_m$ and any $\vec{x} = x_1, \dots, x_m \in M$,

$$A\{\vec{\mathsf{x}} :\equiv \vec{x}\} \equiv A\{\mathsf{x}_1 :\equiv x_1\} \cdots \{\mathsf{x}_m :\equiv x_m\}. \tag{48}$$

For example, in $\mathsf{T}(\mathbf{N}_0)$,

$$\mathsf{S}(\mathsf{S}(\mathsf{v}))\{\mathsf{x} :\equiv 17\} \equiv \mathsf{S}(\mathsf{S}(17)).$$

2A.12. **Term functions** (generalized polynomials). If E is a term and $\vec{\mathsf{x}} \equiv \mathsf{x}_1, \dots, \mathsf{x}_n$ is a list of distinct variables which includes all the variables in E, we set for any $\vec{x} = (x_1, \dots, x_n) \in M^n$,

$$f_E(\vec{x}) = \mathbf{val}_{\mathbf{M}}(E\{\vec{\mathsf{x}} :\equiv \vec{x}\}) \quad (\vec{x} \in A^n)$$

or, with lots of dots rather than vectors,

$$f_E(x_1, \dots, x_n) = \mathbf{val}_{\mathbf{M}}(E\{\mathsf{x}_1 :\equiv x_1, \dots, \mathsf{x}_n :\equiv x_n\}) \quad (x_1, \dots, x_n \in M).$$

The partial function $f_E : M^n \rightharpoonup M$ is the *term function* or *generalized polynomial* defined in $\mathbf{M}$ by E and the list $\vec{\mathsf{x}}$.

For example, in simplified notation on $\mathbf{N}_0$,

for the term $E \equiv \text{if } (\mathsf{x} = 0) \text{ then } SS(\mathsf{x}) \text{ else } Pd(\mathsf{z})$,

$$f_E(x, y, z) = \text{if } (x = 0) \text{ then } 2 \text{ else } z \dot{-} 1;$$

and if F is a field and $a_0, \dots, a_n \in F$, then

$$\text{if } E \equiv a_0 + a_1\mathsf{x} + \cdots + a_n\mathsf{x}^n, \text{ then } f_E(x) = a_0 + a_1 x + \cdots + a_n x^n,$$

which is where the "generalized polynomial" name comes from.

A partial function $f : M^n \rightharpoonup M$ which is definable in this way by a term E and a sequence of variables $\vec{\mathsf{x}}$ is variously called *term definable*, *explicit*, or a *generalized polynomial* of $\mathbf{M}$.

2A.13. **Model theoretic notation**. For any two closed terms, we write

$$\mathbf{M} \models A = B \iff \mathbf{val}_{\mathbf{M}}(A) = \mathbf{val}_{\mathbf{M}}(B) \tag{49}$$
$$\iff \Big(\mathbf{val}_{\mathbf{M}}(A)\!\uparrow \ \& \ \mathbf{val}_{\mathbf{M}}(B)\!\uparrow\Big) \vee (\exists w \in M)\Big(\mathbf{val}_{\mathbf{M}}(A) = \mathbf{val}_{\mathbf{M}}(B) = w\Big).$$

In particular, if $B \equiv w$ is a constant,

$$\mathbf{M} \models A = w \iff \mathbf{val}_{\mathbf{M}}(A) = w.$$

This notation (from logic) avoids the use of subscripts and is sometimes easier to read than the $\mathbf{val}_{\mathbf{M}}(\ )$ notation, especially in the second case, when A is a complex term and $w \in M$.

This notation can also be extended to arbitrary terms A, B which need not be closed: if the list of distinct variables $\vec{\mathsf{x}}$ includes all the variables that occur in A and B, we set

$$\mathbf{M} \models A = B \iff (\forall \vec{x})[f_A(\vec{x}) = f_B(\vec{x})]. \tag{50}$$

For example,

$$\mathbf{N}_0 \models Pd(S(x)) = x$$

is a basic identity of $\mathbf{N}_0$. (And Problem x2A.7 verifies that the choice of the particular list $\vec{\mathsf{x}}$ is irrelevant in this definition.)

Problems for Section 2A

2A.14. A **term derivation** in $\mathsf{T}(\mathbf{M})$ is any finite sequence

$$D = (A_0, \dots, A_n)$$

of words such that for every $j \leq n$ one of the following is true:

(TD1) A_j is a constant $x \in M$.

(TD2) A_j is an individual variable v_i.

(TD3) $A_j \equiv \mathsf{f}_i(E_1, \dots, E_{n_i})$, where f_i is a function symbol with arity n_i and the words $A_1, \dots, A_{n_i}$ occur in $(A_0, \dots, A_{j-1})$, before the j'th place in D.

(TD4) $A_j \equiv (\mathsf{if}\ (E_1 = 0)\ \mathsf{then}\ E_2\ \mathsf{else}\ E_3)$, where the words E_0, E_1, E_2 occur in $(A_0, \dots, A_{j-1})$, before the j'th place in D.

x2A.1. Prove that a word E is a term if and only if there is a term derivation $D = (A_0, \dots, A_n)$ with $E \equiv A_n$.

x2A.2. Prove parts (2) and (3) of Lemma 2A.5.

x2A.3. Prove the Lemma of Unique Readability of Terms 2A.6.

x2A.4 (Unique readability for equations). Prove that if A_1, B_1, A_2, B_2 are terms of $\mathsf{T}(\mathbf{M})$ and

$$A_1 = B_1 \equiv A_2 = B_2$$

then $A_1 \equiv A_2$ and $B_1 \equiv B_2$.

x2A.5. Prove Lemma 2A.7.

x2A.6. State and prove Lemmas similar to 2A.7 which justify recursive definitions on the closed and the pure terms.

x2A.7. Suppose all the individual variables which occur in a term A are in the list $\vec{\mathsf{x}}$ and y is a variable which does not occur in A. Prove that for all $\vec{x}, y \in M$,

$$f_A(\vec{x}) = f_A(\vec{x}, y)$$

x2A.8. Prove that if $\mathbf{M}$ is total, then every generalized polynomial of $\mathbf{M}$ is a total function.

x2A.9. Prove that for every generalized polynomial $f(x_1, \dots, x_n)$ of $\mathbf{N}_0$, there is a number M such that

• Correction 1/20/15

$$f(x_1, \dots, x_n) \leq \max(x_1, \dots, x_n) + M \quad (x_1, \dots, x_n \in \mathbb{N}).$$

Infer that addition $x + y$ is not a generalized polynomial of $\mathbf{N}_0$.

(Intensionally left blank so the paging agrees with the printed version.)

2B. Recursion and computation

We extend here the term language $\mathsf{T}(\mathbf{M})$ associated with a partial algebra $\mathbf{M}$ so that we can give recursive definitions. Our main aim is to define and to prove the basic properties of the fundamental class of (general) **recursive partial functions** in $\mathbf{M}$.

2B.1. **The programming language $\mathsf{R}(\mathbf{M})$: syntax**. The vocabulary of $\mathsf{R}(\mathbf{M})$ extends that of $\mathsf{T}(\mathbf{M})$ by the addition of **function** or **recursive variables**

$$\mathsf{p}_0^n, \mathsf{p}_1^n, \dots \quad (n = 0, 1, \dots, \ \text{arity}(\mathsf{p}_i^n) = n),$$

infinitely many for every arity $n = 0, 1, \dots$.

From the syntactical point of view, the function variables are treated exactly like the function constants $\mathsf{f}_1, \dots, \mathsf{f}_K$ of $\mathsf{T}(\mathbf{M})$: so the terms of $\mathsf{R}(\mathbf{M})$ are defined by the recursion

$$A :\equiv x \mid \mathsf{v}_i \mid \mathsf{f}_i(A_1, \dots, A_{n_i}) \mid \mathsf{p}_i^n(A_1, \dots, A_n) \\ \mid (\mathsf{if}\ (A_1 = 0)\ \mathsf{then}\ A_2\ \mathsf{else}\ A_3)$$

and have all the properties that we have proved—unique readability, etc. They are naturally interpreted in expansions of $\mathbf{M}$: if, for example, all the function variables in a term E are in the list $\vec{\mathsf{p}} \equiv \mathsf{p}_1, \dots, \mathsf{p}_m$, then E can be evaluated in any expansion $(\mathbf{M}, p_1, \dots, p_m)$ which associates a partial function p_i of the appropriate arity with each function variable p_i that occurs in E.

A term of $\mathsf{R}(\mathbf{M})$ is **explicit** if no function variables occurs in it, i.e., if it is a term of $\mathsf{T}(\mathbf{M})$. And as before, a term is **pure** if the only individual constants that (may) occur in it are 0 or 1 and **closed** if no individual variables occur in it.

A **formal recursive equation** is an expression

$$\mathsf{p}(\mathsf{x}_1, \dots, \mathsf{x}_n) = A,$$

in the vocabulary $(\mathsf{f}_1, \dots, \mathsf{f}_K, \mathsf{p}_0^n, \mathsf{p}_1^n, \dots)$ of $\mathsf{R}(\mathbf{M})$ where

- p is a recursive variable or arity n, i.e., $\mathsf{p} \equiv \mathsf{p}_j^n$ for some j;
- $\mathsf{x}_1, \dots, \mathsf{x}_n$ are distinct individual variables; and

- A is a pure term of $\mathsf{R}(\mathbf{M})$ in which no individual variables other than $\mathsf{x}_1, \ldots, \mathsf{x}_n$ occur.

The equation is **explicit** if A is explicit.

For example,

$$\mathsf{p}(x) = \mathsf{if}\ (\mathsf{x} = 0)\ \mathsf{then}\ 1\ \mathsf{else}\ 0$$

is an explicit equation in every partial algebra;

$$\mathsf{p}(\mathsf{x}) = \mathsf{if}\ (\mathsf{x} = 0)\ \mathsf{then}\ 0\ \mathsf{else}\ \mathsf{S}(\mathsf{p}(\mathsf{Pd}(\mathsf{x})))$$

is a recursive (but not explicit) equation of $\mathbf{N}_0$; and

$$\mathsf{p}(\mathsf{x}) = S(\mathsf{y})$$

is not a recursive equation, because the variable y occurs on the right-hand side but not on the left-hand side.

Finally, a **recursive program** of the partial algebra $\mathbf{M}$ is any system of recursive equations

$$(E) \qquad \begin{array}{ll} (e_0) & \mathsf{p}_0(\vec{\mathsf{x}}_0) = E_0 \\ & \vdots \\ (e_k) & \mathsf{p}_k(\vec{\mathsf{x}}_k) = E_k \end{array}$$

where the function variables $\mathsf{p}_0, \ldots, \mathsf{p}_k$ are distinct and they are the only function variables which occur in the terms $E_0, \ldots, E_k$. The equations of E are called (recursive) **definitions** of the function variables $\mathsf{p}_0, \ldots, \mathsf{p}_k$.

The basic idea of the semantics of $\mathsf{R}(\mathbf{M})$ is that each program E assigns (and expresses an algorithm that computes) a partial function

$$\overline{\mathsf{p}}_i : M^{k_i} \rightharpoonup M \quad (\text{if } \mathrm{arity}(\mathsf{p}_i) = k_i)$$

to each recursive variable p_i defined by it, so that $\overline{\mathsf{p}}_0, \ldots, \overline{\mathsf{p}}_k$ *satisfy* E, i.e.,

$$(\mathbf{M}, \overline{\mathsf{p}}_0, \ldots, \overline{\mathsf{p}}_k) \models \mathsf{p}_i(\vec{x}_i) = E_i\{\vec{\mathsf{x}}_i :\equiv \vec{x}_i\} \quad (i = 0, \ldots, k)$$

for all tuples $\vec{x}_i$ from M.

Before defining rigorously the correspondence

$$E \mapsto (\overline{\mathsf{p}}_0, \ldots, \overline{\mathsf{p}}_k)$$

we consider some examples—using "simplified notation", i.e., without worrying about using the correct fonts and spelling out all the terms correctly.

The definition

$$(E1) \qquad \mathsf{p}(x, y) = S(y)$$

is by itself a program of $\mathbf{N}_0$, which defines (explicitly) the binary function variable p. The semantics should obviously give

$$\overline{\mathsf{p}}(x, y) = \mathbf{val}_{\mathbf{N}_0}(S(y)) = y + 1,$$

and this is what they will do.

The equation

$$(E2) \qquad \mathsf{p}(y,\vec{x}) = \text{if } (g(y,\vec{x})=0) \text{ then } y \text{ else } \mathsf{p}(S(y),x)$$

is a program of $(\mathbf{N}_0, g)$ where $g : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$, and there are examples of functions g for which the equation $(E2)$ can have many solutions, Problem x1C.11; for each g however, there exists a least solution of $(E2)$ by Proposition 1C.8, the partial function

$$\overline{\mathsf{p}}(y,\vec{x}) = (\mu i \geq y)[g(i,\vec{x})=0],$$

and this is the solution $\overline{\mathsf{p}}$ that the semantics of $\mathsf{R}(\mathbf{N}_0, g)$ will assign to the symbol p.

Finally, we consider the following trivial example of a program, with the single definition

$$(E_3) \qquad \mathsf{p}(x) = \mathsf{p}(x).$$

Equation (E_3) is satisfied by all partial functions; the semantics will yield the empty partial function

$$\overline{\mathsf{p}}(x) = \varepsilon(x), \text{ i.e., for all } x, \overline{\mathsf{p}}(x)\uparrow,$$

which is, again, the least solution of (E_3), as in example $(E2)$.

After these preliminaries, we proceed to the rigorous definition of the semantics of $\mathsf{R}(\mathbf{M})$. The basic idea is to associate with each recursive program E and each recursive variable p_i of E a **machine** which computes some partial function $\overline{\mathsf{p}}_i$, and we must first make precise what we mean by "machine". We give the definition in two stages.

2B.2. Definition. A **transition system** is any triple

$$\mathcal{T} = (S, \rightarrow, T),$$

where:

(1) S is a non-empty set, the set of *states* of $\mathcal{T}$.
(2) $\rightarrow$ is a binary relation, the *transition relation* on S.
(3) $T \subseteq S$ is the set of *terminal* states of $\mathcal{T}$, and they do not trigger any transitions,

$$(51) \qquad s \in T \Longrightarrow (\forall s')[s \not\rightarrow s'].$$

A state s which satisfies the right-hand side of (51) but is not terminal is *stuck* (or *inactive*). So the states are classified into three categories, terminal, stuck and *active*, that is those that have at least one "next" state. The system $\mathcal{T}$ is **deterministic**, if every state s has at most one successor state, that is

$$(52) \qquad [s \rightarrow s' \ \&\ s \rightarrow s''] \Longrightarrow s' = s''.$$

For an example, set

$$m \to_1 n \iff m > n, \qquad m \to_2 n \iff m = n+1. \tag{53}$$

The system $(\mathbb{N}, \to_1, \{0\})$ is non-deterministic, while the system $(\mathbb{N}, \to_2, \{0\})$ is deterministic.

2B.3. **Computations**. A *partial computation* of a transition system $\mathcal{T}$ is any finite *path* (sequence)

$$Y = (s_0 \to s_1 \to \cdots \to s_n) \tag{54}$$

in the *graph* $(S, \to)$. A partial computation Y is *terminal* or *convergent* if the last state s_n is terminal and *stuck* or *stalled* if s_n is stuck. An *infinite* (divergent) computation is any infinite path

$$Y = (s_0 \to s_1 \to \cdots).$$

The *length* of a finite, partial computation as in (54) is $n+1$.

The transition system $\mathcal{T}$ *computes* the partial function $\pi : S \rightharpoonup T$ on the set of states if, for all $s \in S$ and $t \in T$,

$$\pi(s) = t \iff (\exists (s_0, \dots, s_n) \in C(\mathcal{T}))[s_0 = s \ \&\ s_n = t], \tag{55}$$

where

$$C(\mathcal{T}) = \text{the set of all terminal computations of } \mathcal{T}. \tag{56}$$

It follows that

$$\pi(s)\downarrow \iff (\exists (s_0, \dots, s_n) \in C(\mathcal{T}))[s_0 = s].$$

Non-deterministic transition systems need not compute any partial function on S, but every deterministic system computes exactly one $\pi : S \rightharpoonup T$ which is defined by (55). More than that: if $\mathcal{T}$ is deterministic, then for each state s there exists exactly one terminal, stuck or infinite *complete* computation

$$\mathrm{comp}(s) = \mathrm{comp}_{\mathcal{T}}(s) = (s \to s_1 \to s_2 \to \cdots) \tag{57}$$

which cannot be extended. It is defined by the recursion

$$\begin{aligned} s_0 &= s, \\ s_{n+1} &= \begin{cases} \text{the unique } s' \text{ such that } s_n \to s', & \text{if there is one such } s', \\ \uparrow, & \text{otherwise.} \end{cases} \end{aligned}$$

In order to compute a partial function $f : A \rightharpoonup B$ with a transition system $\mathcal{T}$, we have to enrich $\mathcal{T}$ with some means of "inputting" elements from A and "extracting values" in B.

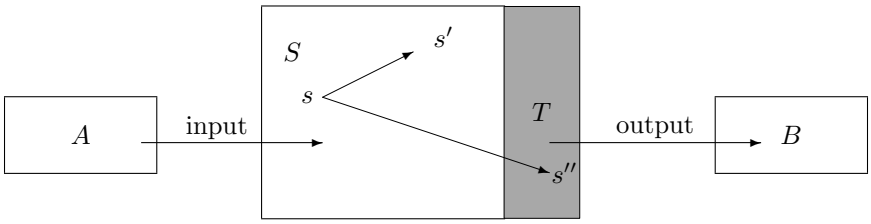


Figure 2. Abstract machine.

2B.4. Definition. An **abstract** or **sequential machine** with *input set* A and *output set* (or *range*) B is any quintuple

$$\mathcal{M} = \mathcal{T}(\text{input}, \text{output}) = (S, \rightarrow, T, \text{input}, \text{output}),$$

where $\mathcal{T} = (S, \rightarrow, T)$ is a transition system and the following additional conditions hold:

(4) $\text{input} : A \to S$ is the *input function* of $\mathcal{M}$.
(5) $\text{output} : T \to B$ is the *output function* of $\mathcal{M}$.

The machine $\mathcal{T}(\text{input}, \text{output})$ computes the partial function $f : A \rightharpoonup B$ if for all $x \in A$, $w \in B$,

(58) $f(x) = w$
$\iff (\exists (s_0, \dots, s_n) \in C(S, \rightarrow, T))[s_0 = \text{input}(x) \ \&\ \text{output}(s_n) = w]$.

A machine is *deterministic* if $\mathcal{T}$ is deterministic, and in this case it computes the partial function $f : A \rightharpoonup B$ which is defined by (58). A nondeterministic machine need not compute a partial function.

2B.5. **Recursive machines**. For each program E of a partial algebra $\mathbf{M}$ we define a transition system $\mathcal{T}(E) = \mathcal{T}(E, \mathbf{M})$ as follows:

(1) The states of $\mathcal{T}(E)$ are all words s of the form

$$\alpha_0 \ \dots \ \alpha_{m-1} : \beta_0 \ \dots \ \beta_{n-1}$$

where the *elements* $\alpha_0, \dots, \alpha_{m_1}, \beta_0, \dots, \beta_{n-1}$ of s satisfy the following conditions:

- each α_i is a function symbol (constant or variable), or a closed term, or the special symbol ?, and
- each β_j is an individual constant, that is $\beta_j \in M$.

Recall that a term A is closed if it does not contain individual variables, and notice that in each state the special symbol ':' has exactly one occurrence.

The states of $\mathcal{T}(E, \mathbf{M})$ are the same for all programs of the partial algebra $\mathbf{M}$ and so we also call them *states of* $\mathbf{M}$. For example, the word

$$\mathsf{p}_1^2\ 3\ \mathsf{S}(3)\ 1\ \mathsf{if}\ \ ?:3\ 0\ 1$$

is a state of $\mathbf{N}_0$, as are the words

$$\mathsf{Pd}\ 1\ 3\ \mathsf{p}_1^2(\mathsf{S}(2)):\qquad :0\ 23$$

or even

$$:$$

This is the simplest state of the recursive machine.

(2) The terminal states of $\mathcal{T}(E)$ are the words of the form

$$: w$$

that is those with no symbols on the left of ':' and only one constant on the right. All the machines based on $\mathcal{T}(E)$ will have a common output function, the function which simply reads this constant,

$$\text{output}(: w) = w.$$

(3) The transition relation of $\mathcal{T}(E)$ is defined by the seven cases in the Transition Table 1. This means $s \to s'$ holds if it is a special case of some line of the Table. Notice that the transitions (e-call) are the only ones which "call" the primitives of $\mathbf{M}$ and so they depend on $\mathbf{M}$, while the transitions (i-call) are the only ones which depend on the specific program E.

The system $\mathcal{T}(E)$ is obviously deterministic.

For each n-place recursive variable p of E, the **recursive machine** $\mathcal{T}(E, \mathsf{p})$ is derived from the transition system $\mathcal{T}(E)$ by the addition of the input function

$$\text{input}(\vec{x}) \equiv \mathsf{p} : \vec{x}$$

and computes the partial function $\overline{\mathsf{p}} = \overline{\mathsf{p}}_E : M^n \rightharpoonup M$, where

$$\overline{\mathsf{p}}_E(\vec{x}) = w \iff \mathsf{p} : \vec{x} \to s_1 \to \cdots \to\ : w. \tag{59}$$

We also use the notation

$$\mathbf{M}, E \vdash \mathsf{p}(\vec{x}) = w \iff \overline{\mathsf{p}}_E(\vec{x}) = w \tag{60}$$

which makes apparent the dependence of $\overline{\mathsf{p}}$ on the partial algebra $\mathbf{M}$. In practice, when the specific program E and the partial algebra $\mathbf{M}$ are obvious from the context, we will simply refer to the partial function $\overline{\mathsf{p}}$.

The **main symbol** of a program E is the function variable p_0 defined by the first equation of E, and E **computes** the partial function $\overline{\mathsf{p}}_0$ in $\mathbf{M}$.

(pass)	$\alpha\ \underline{x:}\ \beta\ \to \alpha\ \underline{:x}\ \beta \quad (x \in M)$
(e-call), in general	$\alpha\ \underline{\mathsf{f}_i : \vec{x}}\ \beta\ \to \alpha\ \underline{:f_i(\vec{x})}\ \beta$
(e-call), for $\mathbf{N}_0$	$\alpha\ \underline{S:x}\ \beta \to \alpha\ \underline{:x+1}\ \beta$ $\alpha\ \underline{Pd:x}\ \beta \to \alpha\ \underline{:x \dot{-} 1}\ \beta$
(i-call)	$\alpha\ \underline{\mathsf{p}_i : \vec{x}}\ \beta\ \to \alpha\ \underline{E_i\{\vec{\mathsf{x}}_i :\equiv \vec{x}\} :}\ \beta$
(comp)	$\alpha\ \underline{h(A_1,\dots,A_n):}\ \beta\ \to \alpha\ \underline{h\ A_1\ \cdots\ A_n :}\ \beta$
(br)	$\alpha\ \underline{(\text{if } (A=0) \text{ then } B \text{ else } C):}\ \beta\ \to \alpha\ \underline{B\ C\ ?\ A:}\ \beta$
(br0)	$\alpha\ \underline{B\ C\ ?:0}\ \beta\ \to \alpha\ \underline{B:}\ \beta$
(br1)	$\alpha\ \underline{B\ C\ ?:y}\ \beta\ \to \alpha\ \underline{C:}\ \beta \quad (y \neq 0)$

- The underlined words are those which change in the transition.
- $\vec{x} = x_1, \dots, x_n$ is an n-tuple of individual constants.
- In the *external call* (e-call), $f = f_i$ is a primitive partial function of $\mathbf{M}$ with $\text{arity}(f_i) = n_i = n$.
- In the *internal call* (i-call), p_i is an n-place recursive variable of the program E which is defined by the equation $\mathsf{p}_i(\vec{\mathsf{x}}) = E_i$.
- In the *composition transition* (comp) h is a function symbol (constant or variable) with $\text{arity}(h) = n$.

Table 1. Transitions of the system $\mathcal{T}(E, \mathbf{M})$.

For example, if one of the equations of E in $\mathbf{N}_0$ is the explicit equation

$$p(x) = S(S(x)),$$

then, for every x, $\overline{p}(x) = x + 2$ by the following computation:

$$p:x\ \to\ S(S(x)):\ \to\ S\ S(x):\ \to\ S\ S\ x:$$
$$\to S\ S\ :\ x \to S\ :\ x+1 \to\ :\ x+2$$

$$
\begin{array}{rcll}
\mathsf{p} & : & 2\ 3 & \text{(i-call)} \\
\text{if } (2=0) \text{ then } 3 \text{ else } S(\mathsf{p}(Pd(2),3)) & : & & \text{(br)} \\
3\ S(\mathsf{p}(Pd(2),3))\ ?\ 2 & : & & \text{(pass)} \\
3\ S(\mathsf{p}(Pd(2),3))\ ? & : & 2 & \text{(br1)} \\
S(\mathsf{p}(Pd(2),3)) & : & & \text{(comp)} \\
S\ \mathsf{p}(Pd(2),3) & : & & \text{(comp)} \\
S\ \mathsf{p}\ Pd(2)\ 3 & : & & \text{(pass)} \\
S\ \mathsf{p}\ Pd(2) & : & 3 & \text{(comp)} \\
S\ \mathsf{p}\ Pd\ 2 & : & 3 & \text{(pass)} \\
S\ \mathsf{p}\ Pd & : & 2\ 3 & \text{(Pd1)} \\
S\ \mathsf{p} & : & 1\ 3 & \text{(i-call)} \\
S \text{ if } (1=0) \text{ then } 3 \text{ else } S(\mathsf{p}(Pd(1),3)) & : & & \text{(br)} \\
S\ 3\ S(\mathsf{p}(Pd(1),3))\ ?\ 1 & : & & \text{(pass)} \\
S\ 3\ S(\mathsf{p}(Pd(1),3))\ ? & : & 1 & \text{(br1)} \\
S\ S(\mathsf{p}(Pd(1),3)) & : & & \text{(comp)} \\
S\ S\ \mathsf{p}(Pd(1),3) & : & & \text{(comp)} \\
S\ S\ \mathsf{p}\ Pd(1)\ 3 & : & & \text{(pass)} \\
S\ S\ \mathsf{p}\ Pd(1) & : & 3 & \text{(comp)} \\
S\ S\ \mathsf{p}\ Pd\ 1 & : & 3 & \text{(pass)} \\
S\ S\ \mathsf{p}\ Pd & : & 1\ 3 & \text{(Pd1)} \\
S\ S\ \mathsf{p} & : & 0\ 3 & \text{(i-call)} \\
S\ S \text{ if } (0=0) \text{ then } 3 \text{ else } S(\mathsf{p}(Pd(0),3)) & : & & \text{(br)} \\
S\ S\ 3\ S(\mathsf{p}(Pd(0),3))\ ?\ 0 & : & & \text{(pass)} \\
S\ S\ 3\ S(\mathsf{p}(Pd(0),3))\ ? & : & 0 & \text{(br0)} \\
S\ S\ 3 & : & & \text{(pass)} \\
S\ S & : & 3 & \text{(S)} \\
S & : & 4 & \text{(S)} \\
 & : & 5 &
\end{array}
$$

FIGURE 3. The computation of $2+3$ by the program $\mathsf{p}(i,x) = \text{if } (i=0) \text{ then } x \text{ else } S(\mathsf{p}(Pd(i),x))$.

For a more interesting example, we observe that addition is defined recursively in $\mathbf{N}_0$ by the program with the single equation

$$p(i,x) = \text{if } (i=0) \text{ then } x \text{ else } S(p(Pd(i),x)).$$

Figure 3 shows the computation of the value $\overline{p}(2,3) = 5$.

2B.6. **M-recursive partial functions, relations and sets**. An n-place partial function $f : M^n \rightharpoonup M$ is **recursive** in the partial algebra $\mathbf{M}$ (or **M-recursive**) if $f = \overline{\mathsf{p}}$ for some program E of $\mathbf{M}$ and some recursive variable p of E, i.e., if

$$f(\vec{x}) = w \iff \mathbf{M}, E \vdash \mathsf{p}(\vec{x}) = w$$

with the notation of (60). Since the order in which we enumerate the equations of a program does not change the definition of the partial functions $\overline{\mathsf{p}}$, *a partial function $f : M^n \rightharpoonup M$ is $\mathbf{M}$-recursive if and only if it is computed by some program of $\mathbf{M}$*, i.e., $f(\vec{x}) = \overline{\mathsf{p}}_0(\vec{x})$ where p_0 is the main symbol of E, Problem x2B.2. We set

$$\mathcal{R}(\mathbf{M}) = \{f : M^n \rightharpoonup M \mid f \text{ is } \mathbf{M}\text{-recursive}\}.$$

A relation $P(\vec{x})$ on M is **M-recursive** if its characteristic function (19) is $\mathbf{M}$-recursive, and a set $A \subseteq M$ is $\mathbf{M}$-recursive if its characteristic function (20) is $\mathbf{M}$-recursive.

For the algebra $\mathbf{N}_0$ and its expansions in which we are especially interested, we will write

$$\mathcal{R} = \mathcal{R}(\mathbb{N}, 0, 1, S, Pd) = \{f : \mathbb{N}^n \rightharpoonup \mathbb{N} \mid f \text{ is } \mathbf{N}_0\text{-recursive}\},$$

and for every set Ψ of partial functions of several variables on $\mathbb{N}$,

$$\mathcal{R}(\Psi) = \{f : \mathbb{N}^n \rightharpoonup \mathbb{N} \mid f \text{ is } (\mathbf{N}_0, f_1, \dots, f_K)\text{-recursive for some } f_1, \dots, f_K \in \Psi\}, \tag{61}$$

so that $\mathcal{R} = \mathcal{R}(\emptyset)$. A partial function on $\mathbb{N}$ is **recursive** if it is $\mathbf{N}_0$-recursive and **recursive in** Ψ if it is $(\mathbf{N}_0, \Psi)$-recursive, and similarly for relations on $\mathbb{N}$ and subsets of $\mathbb{N}$.

Problems for Section 2B

x2B.1. Describe the partial computations of the transition systems (53) and determine the partial functions that they compute.

x2B.2. Prove that a partial function $f(\vec{x})$ is $\mathbf{M}$-recursive if and only if it is computed by some recursive program of $\mathbf{M}$.

x2B.3. For the following three recursive programs in $\mathbf{N}_0$,

$$p(x) = S(p(x)), \tag{E_1}$$

$$p(x) = p(q(x)), \quad q(x) = x, \tag{E_2}$$

$$p(x,y) = p_1(p(x,y), y), \quad p_1(x,y) = x. \tag{E_3}$$

(1) Which partial functions satisfy them, as systems of equations?

(2) Which partial functions do they compute, and how do their computations differ?

x2B.4*. Prove that there exists a unary recursive relation $R(x)$ which is not primitive recursive.

x2B.5. Construct a recursive program of $\mathbf{N}_0$ which computes the Ackermann function, 1A.6.

x2B.6. Construct a recursive program which computes the function $p(x, y)$ in Problem x1C.8*.

x2B.7. Construct a recursive program E in the expansion $(\mathbf{N}_0, g, h, \tau)$ of $\mathbf{N}_0$ which computes the partial function f defined from the functions g, h, τ by nested recursion, Problem x1B.19*.

x2B.8*. Prove that the "equation"

$$f(x) = \begin{cases} 0, & \text{if } g(x)\uparrow, \\ g(x)+1, & \text{otherwise.} \end{cases}$$

"cannot be formalized in R", but first explain what needs to be proved.

x2B.9. Prove or give a counterexample for each of the following two propositions:

(1) For every partial algebra $\mathbf{M}$ and every $x_0 \in M$, the constant, one-place function $f(x) = x_0$ is $\mathbf{M}$-recursive.

(2) For every number $x_0 \in \mathbb{N}$, the constant, one-place function $f(x) = x_0$ on the natural numbers is recursive.

2B.7. Definition. (1) A set $X \subseteq M$ is *closed under the primitives* of a partial algebra $\mathbf{M}$ or **$\mathbf{M}$-closed** if $0, 1 \in X$, and

$$\Big(x_1, \dots, x_{n_i} \in X \text{ and } f_i(x_1, \dots, x_{n_i}) = w\Big) \Longrightarrow w \in X, \quad (i = 1, \dots, K).$$

(2) For each set $A \subseteq M$, set recursively

$$\overline{A}^{(0)} = A \cup \{0, 1\},$$
$$\overline{A}^{(k+1)} = \overline{A}^{(k)} \cup \{f_i(\vec{x}) \mid f_i(\vec{x})\downarrow \ \ \& \ \vec{x} = x_1, \dots, x_{n_i} \in \overline{A}^{(k)}, i = 1, \dots, K\}.$$

The union $\overline{A} = \bigcup_{k=0}^{\infty} \overline{A}^{(k)}$ is the set **generated** by A in $\mathbf{M}$.

x2B.10. Prove that for every partial algebra $\mathbf{M}$ and $A \subseteq M$, the set $\overline{A}$ is the smallest $\mathbf{M}$-closed subset of M which contains A, that is: $A \subseteq \overline{A}$, $\overline{A}$ is $\mathbf{M}$-closed, and for every $X \subseteq M$, if $A \subseteq X$ and X is $\mathbf{M}$-closed, then $\overline{A} \subseteq X$.

x2B.11*. Prove that if $f : M^n \rightharpoonup M$ is $\mathbf{M}$-recursive, then

$$f(x_1, \dots, x_n)\downarrow \implies f(x_1, \dots, x_n) \in \overline{\{x_1, \dots, x_n\}}.$$

Hint: Prove the stronger proposition, that if $A \subseteq M$ and the partial function $f : M^n \rightharpoonup M$ is $\mathbf{M}$-recursive, then the set $\overline{A}$ generated by A is closed under f, that is

$$\Big(x_1, \dots, x_n \in \overline{A} \ \&\ f(x_1, \dots, x_n) = w\Big) \implies w \in \overline{A}.$$

2C. Soundness and least solutions

In the two, fundamental theorems of this section we give a structural characterization of the partial functions computed by the transition systems $\mathcal{T}(E, \mathbf{M})$ which imply the basic properties of the class of $\mathbf{M}$-recursive partial functions.

Key to the proofs is the following, simple

2C.1. Lemma. *For every partial computation*

$$\alpha_0 : \beta_0 \to \alpha_1 : \beta_1 \to \cdots \to \alpha_m : \beta_m$$

in the system $\mathcal{T}(E, \mathbf{M})$ and any words α^*, β^* such that

$$\alpha^*\ \alpha_0 : \beta_0\ \beta^*$$

is a state, the sequence

$$\alpha^*\ \alpha_0 : \beta_0\ \beta^* \to \alpha^*\ \alpha_1 : \beta_1\ \beta^*, \to \cdots \to \alpha^*\ \alpha_m : \beta_m\ \beta^*$$

is also a partial computation of $\mathcal{T}(E, \mathbf{M})$.

It follows that if

$$\alpha_0 : \beta_0 \to \alpha_1 : \beta_1 \to \cdots$$

is a divergent computation and $\alpha^*\ \alpha_0 : \beta_0\ \beta^*$ is a state, then the sequence

$$\alpha^*\ \alpha_0 :\ \beta_0\ \beta^* \to \alpha^*\ \alpha_1 : \beta_1\ \beta^* \to \cdots$$

is also a divergent computation.

Proof is by induction on $m \geq 0$, and it is trivial at the basis $m = 0$, by the hypothesis. At the induction step, we assume that the sequence

$$\alpha^*\ \alpha_0 : \beta_0\ \beta^*\ \to \alpha^*\ \alpha_1 : \beta_1\ \beta^*\ \to \cdots \to \alpha^*\ \alpha_m : \beta_m\ \beta^*$$

is a partial computation, we examine separately the seven cases which justify the transition

$$\alpha_m : \beta_m \to \alpha_{m+1} : \beta_{m+1}$$

in the given computation, and it is obvious that in each of these cases, the same line from Table 1 also justifies the transition

$$\alpha^*\ \alpha_m : \beta_m\ \beta^* \to \alpha^*\ \alpha_{m+1} : \beta_{m+1}\ \beta^*$$

The second conclusion is obtained by applying the first to all the partial computations

$$\alpha_0 : \beta_0 \to \alpha_1 : \beta_1 \to \cdots \to \alpha_m : \beta_m \quad (m \in \mathbb{N}) \qquad \dashv$$

2C.2. Theorem (**Soundness of** $\mathsf{R}(\mathbf{M})$). *For any recursive program E of $\mathbf{M} = (M, 0, 1, f_1, \dots, f_K)$ with recursive variables $\mathsf{p}_0, \dots, \mathsf{p}_k$, let*

$$\overline{\mathbf{M}} = (\mathbf{M}, \overline{\mathsf{p}}_0, \dots, \overline{\mathsf{p}}_k) = (M, 0, 1, f_1, \dots, f_K, \overline{\mathsf{p}}_0, \dots, \overline{\mathsf{p}}_k).$$

It follows that for every closed term A in the vocabulary $(\mathsf{f}_1, \dots, \mathsf{f}_K, \mathsf{p}_0, \dots, \mathsf{p}_k)$:

(1) *If $\mathbf{val}_{\overline{\mathbf{M}}}(A) \uparrow$, then the computation $\mathrm{comp}_{\mathcal{T}}(A :)$ of $\mathcal{T}(E, \mathbf{M})$ with initial state $A :$ is infinite or gets stuck.*

(2) *If $\mathbf{val}_{\overline{\mathbf{M}}}(A) = w$, then the computation $\mathrm{comp}_{\mathcal{T}}(A :)$ of $\mathcal{T}(E, \mathbf{M})$ with initial state $A :$ converges with terminal state $: w$.*

(3) *The partial functions $\overline{\mathsf{p}}_0, \dots, \overline{\mathsf{p}}_k$ satisfy the system E in the partial algebra $\overline{\mathbf{M}}$.*

In particular, every recursive program of $\mathbf{M}$ has solutions.

Proof. For (1) and (2) we use induction on the given, closed term A. We consider cases:

(T1) If $A \equiv x \in M$, then $\mathbf{val}_{\mathbf{M}}(A) = x$, and the computation

$$\begin{aligned} x\ :&\qquad \text{(pass)}\\ :\ x& \end{aligned}$$

yields the correct value.

(T2) If $A \equiv f_i(A_1, \dots, A_{n_i})$ for a primitive f_i of $\mathbf{M}$, then the computation $\mathrm{comp}(A :)$ starts with the transition

$$\begin{aligned} f_i(A_1, \dots, A_{n_i})\ :&\qquad \text{(comp)}\\ f_i\ A_1\ \dots\ A_{n_i}\ :& \end{aligned}$$

We consider three subcases for what happens after this:

(T2a) For some j, $\mathbf{val}_{\overline{\mathbf{M}}}(A_j) \uparrow$, and so $\mathbf{val}_{\overline{\mathbf{M}}}(A) \uparrow$. If j is the *largest* number $\leq n_i$ with this property, then by the choice of j and the induction hypothesis, the computation $\mathrm{comp}(A :)$ starts with the steps

$$\begin{aligned} f_i(A_1, \dots, A_{n_i})\ :&\qquad \text{(comp)}\\ f_i\ A_1\ \dots\ A_{n_i}\ :&\qquad \text{(ind. hyp.)}\\ \vdots& \end{aligned}$$

$$\begin{array}{rll} f_i\ A_1\ \ldots\ A_{n_i-1} & : w_{n_i} & \text{(ind. hyp.)} \\ & \vdots & \\ f_i\ A_1\ \ldots\ A_j & : w_{j+1}\ \cdots\ w_{n_i} & \end{array}$$

By the induction hypothesis again, the computation

$$\mathrm{comp}(A_j :) = A_j : \ \to \alpha_1 : \beta_1 \to \cdots$$

is infinite or gets stuck, since $\mathbf{val}_{\overline{\mathsf{M}}}(A_j)\uparrow$, and by Lemma 2C.1, the computation

$$\begin{array}{l} f_i\ A_1\ \ldots\ A_{j-1}\ A_j : w_{j+1}\ \cdots\ w_{n_i} \\ \qquad\qquad \to f_i\ A_1\ \ldots\ A_{j-1}\ \alpha_1 : \beta_1\ w_{j+1}\ \cdots\ w_{n_i} \to \cdots \end{array}$$

is also infinite or gets stuck, which implies that $\mathrm{comp}(A :)$ has the same property.

(T2b) There are elements $w_1, \ldots, w_{n_i}$ in M such that $\mathbf{val}_{\overline{\mathsf{M}}}(A_j) = w_j$ for $j = 1, \ldots, n_i$, but $f_i(w_1, \ldots, w_{n_i})\uparrow$. In this case, by the induction hypothesis and by appealing again to Lemma 2C.1, the computation $\mathrm{comp}(A :)$ starts with the steps

$$\begin{array}{rll} f_i(A_1, \ldots, A_{n_i}) & : & \text{(comp)} \\ f_i\ A_1\ \ldots\ A_{n_i} & : & \text{(ind. hyp.)} \\ & \vdots & \\ f_i\ A_1\ \ldots\ A_{n_i-1} & : w_{n_i} & \text{(ind. hyp.)} \\ & \vdots & \\ f_i & : w_1\ w_2\ \cdots\ w_{n_i} & \end{array}$$

and here the computation stops (gets stuck) because $f_i(w_1, \ldots, w_{n_i})\uparrow$.

(T2c) $\mathbf{val}_{\overline{\mathsf{M}}}(f_i(A_1, \ldots, A_{n_i})) = w$, so that there are elements $w_1, \ldots, w_{n_i}$ in M with $\mathbf{val}_{\overline{\mathsf{M}}}(A_j) = w_j$ for $j = 1, \ldots, n_i$ and $f_i(w_1, \ldots, w_{n_i}) = w$. From the induction hypothesis and by Lemma 2C.1 once more, the computation $\mathrm{comp}(A :)$ is now as follows:

$$\begin{array}{rll} f_i(A_1, \ldots, A_{n_i}) & : & \text{(comp)} \\ f_i\ A_1\ \ldots\ A_{n_i} & : & \text{(ind. hyp.)} \\ & \vdots & \\ f_i\ A_1\ \ldots\ A_{n_i-1} & : w_{n_i} & \text{(ind. hyp.)} \\ & \vdots & \\ f_i & : w_1\ w_2\ \cdots\ w_{n_i} & \\ & : f_i(w_1, \ldots, w_{n_i}) & \end{array}$$

which is what we needed to prove.

(T3) If $A \equiv \mathsf{p}_i(A_1, \dots, A_{n_i})$ for some n-place function variable p_i of E, then the computation $\mathrm{comp}(A :)$ starts with the transition

$$\begin{aligned} \mathsf{p}_i(A_1, \dots, A_{n_i}) &: \quad \text{(comp)} \\ \mathsf{p}_i\ A_1\ \dots\ A_{n_i} &: \end{aligned}$$

We consider three subcases as in (T2):

(T3a) For some j, $\mathbf{val}_{\overline{\mathsf{M}}}(A_j)\uparrow$, so $\mathbf{val}_{\overline{\mathsf{M}}}(A)\uparrow$.

(T3b) There are elements $w_1, \dots, w_n$ in M such that $\mathbf{val}_{\overline{\mathsf{M}}}(A_j) = w_j$ for $j = 1, \dots, n$, but $\overline{\mathsf{p}}_i(w_1, \dots, w_n)\uparrow$.

(T3c) $\mathbf{val}_{\overline{\mathsf{M}}}(\mathsf{p}_i(A_1, \dots, A_n)) = w$, which means that there are $w_1, \dots, w_n$ in M such that $\mathbf{val}_{\overline{\mathsf{M}}}(A_j) = w_j$ for $j = 1, \dots, n$ and $\overline{\mathsf{p}}_i(w_1, \dots, w_n) = w$.

For case (T3a) the proof is exactly the same as for (T2a). For (T3b) and (T3c), the proofs are slight variants of the ones for (T2b) and (T2c) based on the definition of $\overline{\mathsf{p}}_i$. For (T3c), for example, from the induction hypothesis and by Lemma 2C.1, the computation $\mathrm{comp}(A :)$ is

$$\begin{aligned} \mathsf{p}_i(A_1, \dots, A_n) &: && \text{(comp)} \\ \mathsf{p}_i\ A_1\ \dots\ A_n &: && \text{(ind. hyp.)} \\ &\vdots \\ \mathsf{p}_i\ A_1\ \dots\ A_{n-1} &: w_n && \text{(ind. hyp.)} \\ &\vdots \\ \mathsf{p}_i &: w_1\ w_2\ \cdots\ w_n && \text{(definition of } \overline{\mathsf{p}}_i\text{)} \\ &\vdots \\ &: \overline{\mathsf{p}}_i(w_1, \dots, w_n) \end{aligned}$$

which is what we needed to show.

(T4) $A \equiv (\mathsf{if}\ (A_1 = 0)\ \mathsf{then}\ A_2\ \mathsf{else}\ A_3)$. We leave this for Problem x2C.1.

(3) By the definition of the recursive machine,

$$\overline{\mathsf{p}}_i(\vec{x}) = w \iff \text{there exists a terminal computation } E_i\{\vec{\mathsf{x}} :\equiv \vec{x}\} : \to s_1 \to \cdots \to\ : w,$$

because the computation of $\overline{\mathsf{p}}_i(\vec{x})$ starts with the steps

$$\begin{aligned} \mathsf{p}_i\ &: \vec{x} \quad \text{(i-call)} \\ E_i\{\vec{\mathsf{x}} :\equiv \vec{x}\}\ &: \end{aligned}$$

Now (1) and (2) imply that

$$E_i\{\vec{\mathsf{x}} :\equiv \vec{x}\} : \to s_1 \to \cdots \to\ : w \iff \mathbf{val}_{\overline{\mathsf{M}}}(E_i\{\vec{\mathsf{x}} :\equiv \vec{x}\}) = w,$$

so that

$$\overline{\mathsf{p}}_i(\vec{x}) = w \iff \mathbf{val}_{\overline{\mathsf{M}}}(E_i\{\vec{\mathsf{x}} :\equiv \vec{x}\}) = w$$

which is what we needed to show. ⊣

2C.3. Corolary. *The set of the* $\mathbf{M}$*-recursive partial functions contains the primitive partial functions* $f_1, \dots, f_K$ *of* $\mathbf{M}$*, the projections*

$$P_i^n(x_1, \dots, x_n) = x_i \quad (i = 1, \dots, n)$$

and the constants $C_0(\vec{x}) = 0$ *and* $C_1(\vec{x}) = 1$*, and it is closed under composition and branching.*

Proof. For the given partial functions, we observe that for the program

$$(E_{f_i}) \qquad \mathsf{p}(x_1, \dots, x_{n_i}) = f_i(x_1, \dots, x_{n_i})$$

obviously, $\overline{\mathsf{p}}(\vec{x}_i) = f(\vec{x}_i)$, since $\overline{\mathsf{p}}$ satisfies E_{f_i}. For the projections and the constant functions 0 and 1 we also use the obvious programs with a single equation,

$$\mathsf{p}(\vec{x}) = x_i, \qquad \mathsf{p}(\vec{x}) = 0, \qquad \mathsf{p}(\vec{x}) = 1.$$

For branching, the hypothesis provides programs E_c, E_g and E_h of $\mathbf{M}$ and specific recursive variables c, g and h of these programs, and we have to construct a new program E which defines some "fresh" variable p, so that

$$\overline{\mathsf{p}}_E(\vec{x}) = \text{if } (\overline{c}_{E_c}(\vec{x}) = 0) \text{ then } \overline{g}_{E_g}(\vec{x}) \text{ else } \overline{h}_{E_h}(\vec{x}),$$

where the subscripts suggest the programs which compute $\overline{c}_{E_c}$, $\overline{g}_{E_g}$ and $\overline{h}_{E_h}$. Using alphabetic variants (for the recursive variables) of E_c, E_g, E_h, we can assume that these programs have no common function variables, and we set

$$E = E_c + E_g + E_h + \{\mathsf{p}(\vec{x}) = \text{if } (c(\vec{x}) = 0) \text{ then } g(\vec{x}) \text{ else } h(\vec{x})\},$$

where by "+" we mean the collection of all definitions in the given programs. We note that E is a program, since every recursive variable in it is defined exactly once. Moreover,

$$\overline{c}_E(\vec{x}) = \overline{c}_{E_c}(\vec{x}),$$

just because every computation

$$\mathrm{comp}_{E_c}(c : \vec{x}) = c : \vec{x} \to \alpha_1 : \beta_1 \to \cdots$$

of E_c is also a computation of E, and so (by the determinism of programs) it is the only computation in E which starts with the state $c : \vec{x}$, that is

$$\mathrm{comp}_{E_c}(c : \vec{x}) = \mathrm{comp}_E(c : \vec{x});$$

so $\overline{c}_E = \overline{c}_{E_c}$, and the same, of course, holds also for the symbols g and h. Finally, by Theorem 2C.2, the function $\overline{\mathsf{p}}_E$ satisfies the equation which defines it in E, and so

$$\begin{aligned}\overline{\mathsf{p}}(\vec{x}) &= \text{if } (\overline{c}_E(\vec{x}) = 0) \text{ then } \overline{g}_E(\vec{x}) \text{ else } \overline{h}_E(\vec{x})\\ &= \text{if } (\overline{c}_{E_c}(\vec{x}) = 0) \text{ then } \overline{g}_{E_g}(\vec{x}) \text{ else } \overline{h}_{E_h}(\vec{x}).\end{aligned}$$

The proof for composition is similar, Problem x2C.3. ⊣

2C.4. Corolary. *For each partial algebra* $\mathbf{M} = (M, 0, 1, f_1, \dots, f_K)$ *and any* $g : M^n \rightharpoonup M$, $f : M^m \rightharpoonup M$,

$$\Big(g \in \mathcal{R}(\mathbf{M}) \ \&\ f \in \mathcal{R}(\mathbf{M}, g)\Big) \Longrightarrow f \in \mathcal{R}(\mathbf{M}),$$

where $(\mathbf{M}, g) = (M, 0, 1, f_1, \dots, f_K, g)$ *is the expansion of* $\mathbf{M}$ *by* g.

Proof. Problem x2C.4. ⊣

The next theorem gives a characterization of the *canonical solutions* of a program E computed by the recursive machine:

2C.5. Theorem (**Least Fixed Points**). *For any program* E *of a partial algebra* $\mathbf{M}$ *with recursive variables* $\mathsf{p}_0, \dots, \mathsf{p}_k$, *the partial functions* $\overline{\mathsf{p}}_0, \dots, \overline{\mathsf{p}}_k$ *computed by* $\mathcal{T}(E)$ *are the* $\sqsubseteq$*-least partial functions which satisfy the equations of* E.

Proof. The functions $\overline{\mathsf{p}}_0, \dots, \overline{\mathsf{p}}_k$ satisfy the system E by Theorem 2C.2, so it suffices to show that *if* $\mathsf{p}'_0, \dots, \mathsf{p}'_k$ *also satisfy the equations of* E, *then*

$$\overline{\mathsf{p}}_i(\vec{x}) = w \Longrightarrow \mathsf{p}'_i(\vec{x}) = w \quad (i = 0, \dots, k).$$

So we assume that the functions $\mathsf{p}'_0, \dots, \mathsf{p}'_k$ satisfy the equations of E, and we consider the two structures

$$\overline{\mathbf{M}} = (\mathbf{M}, \overline{\mathsf{p}}_0, \dots, \overline{\mathsf{p}}_k), \qquad \mathbf{M}' = (\mathbf{M}, \mathsf{p}'_0, \dots, \mathsf{p}'_k).$$

By Theorem 2C.2, we know that for each closed term A in the vocabulary $(f_1, \dots, f_K, \mathsf{p}_0, \dots, \mathsf{p}_k)$, if $\mathbf{val}_{\overline{\mathbf{M}}}(A) = w$, then the computation $\mathrm{comp}(A :)$ of $\mathcal{T}(E)$ is terminal with $: w$ its last state. We will show by induction on m, that for each A and every w,

$$\text{if } A : \rightarrow \alpha_1 : \beta_1 \rightarrow \cdots \alpha_{m-1} : \beta_{m-1} \rightarrow \ : w, \text{ then } \mathbf{val}_{\mathbf{M}'}(A) = w. \tag{62}$$

In the special case $A \equiv \mathsf{p}_i(x_1, \dots, x_n)$, this yields

$$\overline{\mathsf{p}}_i(\vec{x}) = w \Longrightarrow \mathbf{val}_{\mathbf{M}'}(\mathsf{p}_i(\vec{x})) = w \Longrightarrow \mathsf{p}'_i(\vec{x}) = w,$$

which is what we need to show.

For the proof of (62) we examine the form of A, and the argument is trivial (as in the proof of 2C.2) in all the cases except when

$$A \equiv \mathsf{p}_i(A_1, \dots, A_n).$$

In this case the computation has the form

$$\begin{aligned}
\mathsf{p}_i(A_1,\dots,A_n) &: \\
\mathsf{p}_i\ A_1\ \cdots\ A_n &: \\
&\vdots \\
\mathsf{p}_i\ A_1\ \cdots\ A_{n-1} &: w_n \\
&\vdots \\
\mathsf{p}_i &: w_1\ \cdots\ w_n \\
E_i\{\vec{\mathsf{x}} :\equiv \vec{w}\} &: \\
&\vdots \\
&: w
\end{aligned}$$

Now the induction hypothesis implies that

$$\mathbf{val}_{\mathbf{M}'}(A_1) = w_1,\dots,\mathbf{val}_{\mathbf{M}'}(A_n) = w_n, \mathbf{val}_{\mathbf{M}'}(E_i\{\vec{\mathsf{x}} :\equiv \vec{w}\}) = w$$

because the computations which correspond to these values have smaller length. So

$$\begin{aligned}
\mathbf{val}_{\mathbf{M}'}(\mathsf{p}_i(A_1,\dots,A_n)) &= \mathsf{p}_i'(\mathbf{val}_{\mathbf{M}'}(A_1),\dots,\mathbf{val}_{\mathbf{M}'}(A_n)) \\
&= \mathsf{p}_i'(w_1,\dots,w_n) \\
&= \mathbf{val}_{\mathbf{M}'}(E_i\{\vec{\mathsf{x}} :\equiv \vec{w}\}) = w,
\end{aligned}$$

where the last equation because by the hypothesis

$$\mathbf{M}' \models \mathsf{p}_i(\vec{\mathsf{x}}) = E_i. \qquad \dashv$$

Problems for Section 2C

x2C.1. Give the missing argument for case (T4) in the proof of Theorem 2C.2. Hint: It is very much like the argument for case (T2), only simpler.

x2C.2*. (1) Prove that for each program E in a *total* algebra $\mathbf{M}$, and each n-place recursive variable p if E, no computation of the form

(*) $$\mathsf{p} : x_1,\dots,x_n \to s_1 \to \cdots \to s_m.$$

is stuck, cf. 2B.3.

(2) Prove that if $\mathbf{M}$ is a partial algebra, p is an n-place function variable of some program E in $\mathbf{M}$ and the finite computation (*) is stuck, then its last state is of the form

$$\alpha\ f_i : y_1,\dots,y_{n_i}\ \beta$$

where f_i is one of the given partial functions of $\mathbf{M}$, and $f_i(y_1, \dots, y_{n_i})\uparrow$.

Hint: Use the method of proof of Theorem 2C.2.

x2C.3. Prove that the composition (10) of $\mathbf{M}$-recursive partial functions is $\mathbf{M}$-recursive.

x2C.4. Prove Corollary 2C.4.

x2C.5. Prove that if $P(\vec{x})$ and $Q(\vec{x})$ are recursive relations on the partial algebra $\mathbf{M}$, then the relations

$$\begin{aligned} R_1(\vec{x}) &\iff \neg P(\vec{x}) \\ R_2(\vec{x}) &\iff P(\vec{x}) \ \& \ Q(\vec{x}), \\ R_3(\vec{x}) &\iff P(\vec{x}) \vee Q(\vec{x}). \end{aligned}$$

are also $\mathbf{M}$-recursive.

x2C.6. Prove that the union $A \cup B$, the intersection $A \cap B$, and the difference

$$A \setminus B = \{x \in A \mid x \notin B\}$$

of two $\mathbf{M}$-recursive sets are $\mathbf{M}$-recursive sets.

Prove also that the singleton $\{0\}$ is $\mathbf{M}$-recursive in every partial algebra $\mathbf{M}$, while in some partial algebra, the singleton $\{1\}$ is not recursive.

2C.6. Definition. For each $f : M \rightharpoonup M$, the **iteration** $f : \mathbb{N} \times M \rightharpoonup M$ of f is defined by the recursion

$$f^0(x) = x, \quad f^{n+1}(x) = f(f^n(x)).$$

x2C.7*. Prove that if $g, h : M \rightharpoonup M$ are $\mathbf{M}$-recursive partial functions, then the function

$$f(x) = g^m(x) \text{ where } m = (\mu n \geq 1)[h^n(x) = 0].$$

is also $\mathbf{M}$-recursive.

2D. Recursive partial functions on the natural numbers

Recall definition (61)

$$\mathcal{R}(\Psi) = \{f : \mathbb{N}^n \rightharpoonup \mathbb{N} \mid f \text{ is } (\mathbf{N}_0, f_1, \dots, f_K)\text{-recursive for some } f_1, \dots, f_K \in \Psi\}$$

of the set of partial functions which are recursive in a set Ψ of partial functions of several variables on the natural numbers. We collect in one theorem the basic properties of $\mathcal{R}(\Psi)$ which follow from the results in the two previous sections. These also hold, of course, for the set

$$\mathcal{R} = \mathcal{R}(\emptyset) = \mathcal{R}(\mathbf{N}_0)$$

of the (absolutely) recursive, partial functions on $\mathbb{N}$.

2D.1. Theorem. *The set $\mathcal{R}(\Psi)$ is primitively closed and closed under minimalization, so that*

$$\mathcal{R}_\mu(\Psi) \subseteq \mathcal{R}(\Psi).$$

In particular, every μ-recursive partial function is recursive.

Proof. The set $\mathcal{R}(\Psi)$ contains the projections, the constants 0 and 1 and the givens $S(x)$ and $Pd(x)$ since every $\mathcal{R}(\mathbf{N}_0, 0, 1, f_1, \dots, f_k)$ with $f_1, \dots, f_k \in \Psi$ has these properties by Corollary 2C.3.

We leave the primitive closure of $\mathcal{R}(\Psi)$ as a problem, x2D.1.

To show show that $\mathcal{R}(\Psi)$ is also closed under minimalization, suppose $g \in \mathcal{R}(\Psi)$ and

$$f(y, \vec{x}) = (\mu i \geq y)[g(i, \vec{x}) = 0].$$

By Proposition 1C.8, the function f is the least solution of the recursive equation

$$f(y, \vec{x}) = \text{if } (g(y, \vec{x}) = 0) \text{ then } y \text{ else } f(y + 1, \vec{x}).$$

This is a program by itself, so (by Theorem 2C.5), f is computed by this program and $f \in \mathcal{R}(\Psi \cup \{g\})$; but if $g \in \mathcal{R}(\Psi)$, then $\mathcal{R}(\Psi \cup \{g\}) = \mathcal{R}(\Psi)$ by Corollary 2C.4. ⊣

Problems for Section 2D

x2D.1. Prove that if $g(\vec{x})$ and $h(w, y, \vec{x})$ are recursive in Ψ and the function $f(y, \vec{x})$ is defined from them by primitive recursion (11), then $f(y, \vec{x})$ is also recursive in Ψ.

Leat

$$\mathbf{N}_b = (\mathbb{N}, 0, 1, \text{Parity}, \text{iq}_2, \text{em}_2, \text{om}_2) \tag{63}$$

where $\text{Parity}(x)$ is 0 or 1 accordingly, if x is even or odd, and

$$\text{iq}_2(x) = \text{quot}(x, 2), \quad \text{em}_2(x) = 2x, \quad \text{om}_2(x) = 2x + 1.$$

The primitives of $\mathbf{N}_b$ are most natural when we want to represent natural numbers by their **binary expansion**,

$$x = x_0 + x_1 2 + x_2 2^2 + \cdots + x_k 2^k \quad (x_i \leq 1).$$

Recall also the "standard" structure $\mathbf{N} = (\mathbb{N}, 0, 1, +, \cdot)$ defined in (44).

x2D.2. Prove that $\mathcal{R}(\mathbf{N}_0) = \mathcal{R}(\mathbf{N}_b) = \mathcal{R}(\mathbf{N})$.

x2D.3*. Prove that the only total solution of the equation $(*)$ in Problem x1C.13* is recursive, but the equation $(*)$ also has solutions which are not recursive partial functions.

CHAPTER 3

COMPUTABILITY AND UNSOLVABILITY

From the central results in this Chapter stem the most important applications of recursion theory on the natural numbers which we will develop in the sequel. We will lay the necessary mathematical foundations for these applications, and we will establish the relationship between "recursion" and "computability" — the famous *Church-Turing Thesis*.

3A. Normal form and enumeration

The main result of this Section is the following,

3A.1. **Normal Form and Enumeration Theorem** [Kleene]. *There is a primitive recursive function $U(y)$, and primitive recursive relations $T_n(e, x_1, \dots, x_n, y)$ $(n \geq 1)$ and functions $S_n^m(e, z_1, \dots, z_m)$ $(n, m \geq 1)$ with the following properties:*

(1) *An n-place partial function $f(\vec{x})$ on the natural numbers is recursive if and only if there is some number e* (a *code* of f) *such that*

$$f(\vec{x}) = U(\mu y T_n(e, \vec{x}, y)) \quad (\vec{x} = x_1, \dots, x_n \in \mathbb{N}). \tag{64}$$

(2) (The S_n^m-Theorem). *For all e, y, $\vec{z} = z_1, \dots, z_m$ and $\vec{x} = x_1, \dots, x_n$,*

$$U(\mu y T_{m+n}(e, \vec{z}, \vec{x}, y)) = U(\mu y T_n(S_n^m(e, \vec{z}), \vec{x}, y)). \tag{65}$$

Moreover, the functions $S_n^m(e, \vec{z})$ are one-to one.

From the *normal form* (64) it follows that every recursive partial function can be defined using primitive recursive functions and a single dumb search, and in particular, every recursive partial function is μ-recursive. In addition, if we set

$$\varphi_e^n(\vec{x}) = U(\mu y T_n(e, \vec{x}, y)), \tag{66}$$

then equation (64) implies that, for each n, the sequence

$$\varphi_0^n, \varphi_1^n, \dots,$$

enumerates all n-place recursive partial functions, in such a manner that the $(n+1)$-place partial function

$$\varphi^n(e,\vec{x}) = \varphi^n_e(\vec{x}) = U(\mu y T_n(e,\vec{x},y))$$

is recursive. The number e is called a **code** or *Gödel number* of the partial function φ^n_e.

The basic idea for the proof is to code in $\mathbb{N}$ (as in 1B.11) the recursive programs of $\mathbf{N}_0$ and the terminal computations of recursive machines, so that the following two conditions hold:

(1) The *basic computation relation*

(67) $T_n(e,x_1,\dots,x_n,y) \iff$ e is the code of a recursive program E and y is the code of a terminal computation of E with input $\mathsf{p}_0 : x_1,\dots,x_n$

is primitive recursive.

(2) There is a primitive recursive function $U(y)$, such that if y is the code of a terminal computation Y, then

(68) $U(y) =$ the output value of Y.

If we can do this, then (1) in Theorem 3A.1 follows immediately and its meaning becomes clear. The significance of the more technical Part (2) will be explained in the sequel.

In the manipulations with primitive recursive functions and relations that we need to make, we will repeatedly use sequence codes defined in 1B.12. To simplify some formulas, we set

$$(u)_{i,j} = ((u)_i)_j, \quad (u)_{i,j,k} = ((u)_i)_j)_k, \text{ etc.},$$
$$\mathrm{first}(u) = (u)_0, \quad \mathrm{last}(u) = (u)_{\mathrm{lh}(u)\dot{-}1},$$

so that when $n > 0$,

$$\mathrm{first}(\langle u_0,\dots,u_{n-1}\rangle) = u_0 = \text{the first element of } (u_0,\dots,u_{n-1}),$$
$$\mathrm{last}(\langle u_0,\dots,u_{n-1}\rangle) = u_{n-1} = \text{the last element of } (u_0,\dots,u_{n-1}).$$

For example

$$(\langle\langle 0,2\rangle,\langle 1,0\rangle\rangle)_{0,1} = 2,\ (\langle\langle 0,2\rangle,\langle 1,0\rangle\rangle)_{1,0} = 1.$$

We will also need some technical properties of sequence codes, including the following:

3A.2. Lemma. *For the classical coding* (1B.13)*, put*

(69) $$\mathrm{seg}(u,i,j) = \begin{cases} \langle (u)_i,(u)_{i+1},\dots,(u)_{j\dot{-}1}\rangle, & \text{if } \mathrm{Seq}(u)\ \&\ 0 \le i < j \le \mathrm{lh}(u), \\ 0, & \text{otherwise.} \end{cases}$$

The function $\mathrm{seg}(u,i,j)$ *is primitive recursive,*

$$\mathrm{seg}(u,i,j) \leq u,$$

and if $i > 0$ *or* $j < \mathrm{lh}(u)$, *then* $\mathrm{seg}(u,i,j) < u$.

Proof in Problem x3A.1. ⊣

The proof of Theorem 3A.1 requires a sequence of definitions and lemmas which provide primitive recursive codings for the basic sets of objects of the theory of the programming language $\mathsf{R} = \mathsf{R}(\mathbf{N}_0)$. More specifically, we will define successively injections

$$[\ ]_i : A_i \rightarrowtail \mathbb{N}$$

for each of six basic sets associated with recursive machines and their computations, and we will prove that various *decoding functions and relations* which extract information about the objects in these sets from their codes are primitive recursive.

1. **Symbols**. For the individual variables, the numbers, the primitives and the recursive function variables, we set first

$$[\mathsf{v}_i]_1 = \langle 0,0,i\rangle, \quad [n]_1 = \langle 0,1,n\rangle, \quad [\mathsf{S}]_1 = \langle 1,1,0\rangle,$$
$$[\mathsf{Pd}]_1 = \langle 1,1,1\rangle, \quad [\mathsf{p}_i^n]_1 = \langle 1,n,2+i\rangle;$$

and for the remaining eight symbols

$$\mathsf{if} \quad \mathsf{then} \quad \mathsf{else} \quad , \quad (\quad) \quad = \quad ?$$

that are used by the recursive machines for $\mathbf{N}_0$, we use the codes

$$\langle 2,0\rangle, \ldots, \langle 2,7\rangle,$$

i.e., $[\mathsf{if}\ ]_1 = \langle 2,0\rangle$, $[\mathsf{then}\ ]_1 = \langle 2,1\rangle$, $\ldots$, $[?]_1 = \langle 2,7\rangle$.

Notice that the code of a number n is much greater than n, e.g.,

$$[1]_1 = \langle 0,1,1\rangle = 2 \cdot 3^2 \cdot 5^2 = 450.$$

Quite obviously, we are not concerned here with the efficiency of the coding.

2. **Words**. For every word

$$\alpha \equiv \alpha_0\alpha_1 \cdots \alpha_n$$

from these symbols, we set

$$[\alpha_0\alpha_1 \cdots \alpha_n]_2 = \langle [\alpha_0]_1, [\alpha_1]_1, \ldots, [\alpha_n]_1\rangle.$$

For example,

$$[\mathsf{S}(\mathsf{v}_1)]_2 = \langle [\mathsf{S}]_1, [(]_1, [\mathsf{v}_1]_1, [)]_1\rangle = \langle\langle 1,1,0\rangle, \langle 2,4\rangle, \langle 0,0,1\rangle, \langle 2,5\rangle\rangle,$$
$$[\mathsf{p}_1^2(\mathsf{v}_1,0)]_2 = \langle [\mathsf{p}_1^2]_1, [(]_1, [\mathsf{v}_1]_1, [,], [0]_1, [)]_1\rangle = \cdots.$$

In particular, this definition assigns codes to the *terms* of $\mathsf{R}(\mathbf{N}_0)$, which are among these words.

Remark. The number variables v_i and the number constants n are symbols, but they are also terms of length 1. So they have two different codings, as symbols and as terms, and we must be careful not to confuse them:

$$[\mathsf{v}_i]_1 = \langle 0, 0, i\rangle, \quad [\mathsf{v}_i]_2 = \langle\langle 0, 0, i\rangle\rangle$$
$$[n]_1 = \langle 0, 1, n\rangle, \quad [n]_2 = \langle\langle 0, 1, n\rangle\rangle.$$

For example, $[0]_1 = \langle 0, 1, 0\rangle = 2 \cdot 3^2 \cdot 5 = 90$, while $[0]_2 = \langle 90\rangle = 2^{91}$.

3. **Recursive equations**. A (formal) equation is determined by its two sides, which are terms, and we set

$$[\mathsf{p}(\vec{\mathsf{v}}) = E]_3 = \langle [\mathsf{p}(\vec{\mathsf{v}})]_2, [E]_2\rangle.$$

4. **Recursive programs**. If $E = (e_0, \dots, e_k)$ is a program, then each e_i is an equation. We set

$$[E]_4 = \langle [e_0]_3, \dots, [e_k]_3\rangle.$$

5. **States**. The elements on the left-hand side of a state are either closed terms or function symbols or the symbol '?', while the elements on its right-hand side are constants, i.e., numbers. We set:

$$[\alpha_0, \dots, \alpha_{m-1} : \beta_0, \dots, \beta_{n-1}]_5 = \langle\langle [\alpha_0]', \dots, [\alpha_{m-1}]'\rangle, \langle [\beta_0]_1, \dots, [\beta_{n-1}]_1\rangle\rangle,$$

where

$$[\alpha_i]' = \begin{cases} [\alpha_i]_1, & \text{if } \alpha_i \text{ is a function symbol or '?'}, \\ [\alpha_i]_2, & \text{otherwise, i.e., if } \alpha_i \text{ is a closed term.} \end{cases}$$

Again, we need to be careful, because the number constants are coded as symbols on the right and as terms on the left-hand side; for example,

$$[2 : 2]_5 = \langle\langle\langle 0, 1, 2\rangle\rangle\rangle, \langle\langle 0, 1, 2\rangle\rangle\rangle = \text{ some huge number.}$$

We notice that when u is the code of a state $\alpha : \beta$, then $\mathrm{first}(u)$ is the code of the left-hand side α, and $\mathrm{last}(u)$ is the code of the right-hand side β.

To verify that the function $\alpha \mapsto [\alpha]_5$ is one-to-one on the set of states, we note that no number is at the same time the code of a function symbol or '?' and also the code of a closed term; because the first symbol of each closed term E is one of

$$\mathsf{S},\ \mathsf{Pd},\ n,\ \mathsf{p}_i^n,\ ($$

with code > 2, so that for every term E, $\mathrm{first}([E]_2) > 2$, while if u is the code of a symbol, then $\mathrm{first}(u) \leq 2$.

6. Computations. For each sequence of states $s = (s_0, \dots, s_k)$, we set

$$[s_0, \dots, s_k]_6 = \langle [s_0]_5, \dots, [s_k]_5 \rangle.$$

This codes all sequences of states, and in particular the terminal computations of the machine for any recursive program.

At this point definitions (67) and (68) have been made precise, and for Part (1) of the theorem it suffices to show that the relation $T_n(e, \vec{x}, y)$ and the function $U(y)$ are primitive recursive.

The second task is easy by a careful reading of the definitions, which shows that we can simply set

$$U(y) = \text{last}(y)_{1,0,2} \tag{70}$$

(Problem x3A.3).

The first task involves a substantial amount of computation. We will show that $T_n(e, \vec{x}, y)$ is primitive recursive in a sequence of three lemmas which introduce codings for several auxilliary sets and establish their basic properties.

3A.3. Lemma. *The following relations and functions are primitive recursive:*

$$\begin{aligned}
\text{IndVar}(v) &\iff v \text{ is the code of some } \mathsf{v}_i \\
&\iff v = \langle 0, 0, (v)_2 \rangle \\
\text{IndConst}(c) &\iff c \text{ is the code of a number} \\
&\iff c = \langle 0, 1, (c)_2 \rangle \\
\text{FunVar}(f) &\iff f \text{ is the code of some } \mathsf{p}_i^n \\
&\iff f = \langle 1, (f)_1, (f)_2 \rangle \ \&\ (f)_1 \geq 1 \ \&\ (f)_2 \geq 2 \\
\text{FunConst}(f) &\iff f \text{ is the code of } \mathsf{S} \text{ or of } \mathsf{Pd} \\
&\iff f = [\mathsf{S}]_1 \vee f = [\mathsf{Pd}]_1 \\
\text{FunSymb}(f) &\iff \text{FunVar}(f) \vee \text{FunConst}(f) \\
\text{arity}(f) &= \text{the arity of the function symbol } f \\
&\quad \text{(if } f \text{ is the code of a function symbol)} \\
&= (f)_1
\end{aligned}$$

Proof is trivial. ⊣

3A.4. Lemma. *The relation*

$$\text{Term}(u) \iff u \text{ is the code of a term}$$

is primitive recursive.

PROOF. By 2A.14, a *term derivation* is a sequence of words

$$D = (A_0, \dots, A_n)$$

which satisfies certain conditions (TD1) – (TD4) that correspond to the conditions in the definition of terms 2A.3. We code term derivations in the obvious way, as sequences of words

$$[A_0, \dots, A_n] = \langle [A_0]_2, \dots, [A_n]_2 \rangle,$$

and we check first that the relation

$$\mathrm{TermDer}(d) \iff d \text{ is the code of a term derivation}$$

is primitive recursive. This is because we can verify whether d is a code of a term derivation by examining its components $(d)_j, (d)_{j,i}$ etc. and using bounded quantifications. Looking at (TD1) – (TD4) in 2A.14, the precise equivalence we need is

$$\begin{aligned}\mathrm{TermDer}(d) \iff{}& \mathrm{Seq}(d) \ \&\ (\forall j < \mathrm{lh}(d))\mathrm{Seq}((d)_j)\\ &\&\ (\forall j < \mathrm{lh}(d))\Big(R_1(d,j) \vee R_2(d,j) \vee R_3(d,j) \vee R_4(d,j)\Big)\end{aligned}$$

where

$$\begin{aligned}R_1(d,j) \iff{}& A_j \equiv c \text{ for some constant } c \in \mathbb{N}\\ \iff{}& (d)_j = \langle (d)_{j,0} \rangle \ \&\ \mathrm{IndConst}((d)_{j,0}),\end{aligned}$$

$$\begin{aligned}R_2(d,j) \iff{}& A_j \equiv \mathsf{v}_i \text{ for some individual variable } \mathsf{v}_i\\ \iff{}& (d)_j = \langle (d)_{j,0} \rangle \ \&\ \mathrm{IndVar}((d)_{j,0})\end{aligned}$$

$$\begin{aligned}R_3(d,j) \iff{}& A_j \equiv \mathsf{f}_i(B_1, \dots, B_{n_i}) \text{ with } B_1, \dots, B_{n_i} \text{ earlier in } D\\ \iff{}& \mathrm{FunSymb}((d)_{j,0})) \ \&\ (d)_{j,1} = [(]_1 \ \&\ (d)_{j,\mathrm{lh}((d)_j) \dot{-} 1} = [)]_1\\ &\&\ \mathrm{lh}((d)_j) = \mathrm{arity}((d)_{j,0}) + 3\\ &\&\ (\forall i < \mathrm{arity}((d)_{j,0}))(\exists k < j)[(d)_{j,i+2} = (d)_k]\end{aligned}$$

$$\begin{aligned}R_4(d,j) \iff{}& A_j \equiv (\mathsf{if}\ (B_1 = 0)\ \mathsf{then}\ B_2\ \mathsf{else}\ B_3)\\ &\text{with } B_1, B_2.B_3 \text{ earlier in } D\\ \iff{}& (\text{a clause similar to that for } R_3, \text{ only simpler, Problem x3A.4})\end{aligned}$$

From this and Problem x2A.1 it follows that

$$\mathrm{Term}(u) \iff (\exists d)[\mathrm{TermDer}(d) \ \&\ u = (d)_{\mathrm{lh}(d) \dot{-} 1}]. \tag{71}$$

This, however, does not suffice to prove that $\mathrm{Term}(u)$ is primitive recursive: we need to find a bound for d in this equivalence, and this comes from the following, improved version of Problem x2A.1:

Sublemma. A word E is a term if and only if there is a term derivation $D = (A_1, \dots, A_n)$ with $E \equiv A_n$ such that:

(1) *every word A_i which occurs in D is a subterm of E.*
(2) *no word occurs twice in D,*

$$i < j \leq n \Longrightarrow A_i \not\equiv A_j.$$

Proof of the Sublemma. The implication ($\Leftarrow$) does not need the extra hypotheses (1) and (2): for any term derivation $D = (A_0, \dots, A_n)$ of E, we simply check by induction on $j \leq n$ that each A_j is a term, and in particular, $E \equiv A_n$ is a term.

The implication ($\Rightarrow$) is proved by induction on terms. For example, in the most complex case, if $E \equiv \mathsf{f}(E_1, \dots, E_k)$ for a function symbol f, the induction hypothesis gives us "nice" derivations $D^1, \dots, D^k$ of each of the terms $E_1, \dots, E_k$, and then lining these up and placing E at the end produces a sequence

$$D = A^1_0, \dots, E_1, A^2_0, \dots, E_2, \dots, A^k_0, \dots, E_k, \mathsf{f}(E_1, \dots, E_k)$$

which is a term derivation of E. It clearly satisfies (1) in the Sublemma, since every term which occurs in it is a subterm of some E_i, and hence a subterm of $E \equiv \mathsf{f}(E_1, \dots, E_k)$. Now, the point is that *if we successively delete from D every term A^i_j which occurs earlier in D*, the resulting sequence is still a term derivation which ends with E and no word occurs twice in it, as required. $\dashv$ (Sublemma)

Suppose now that $D = (A_0 \dots, A_n)$ is a term derivation of some term $E \equiv A_n$ which satisfies (1) and (2) of the Sublemma and $[E]_2 = u$. If $m = \mathrm{lh}(u)$ is the length of the term E, then obviously (and coarsely) $n + 1 \leq m^2$; because each A_i is a piece of E and there are fewer than m^2 choices of a point in E where it starts and another point when it ends. Also, $[A_i]_2 \leq u$ by Lemma 3A.2; so

$$[D] = [A_0, \dots, A_n] \leq u^{m^2} = u^{\mathrm{lh}(u)^2} \leq u^{(u^2)}$$

and hence (71) can be rewritten in the form

$$\mathrm{Term}(u) \iff (\exists d \leq u^{(u^2)})[\mathrm{TermDer}(d) \ \&\ u = (d)_{\mathrm{lh}(d) \dot{-} 1}].$$

The function $(u \mapsto u^{(u^2)})$ is primitive recursive, and so this equivalence completes the proof that $\mathrm{Term}(u)$ is primitive recursive. $\dashv$

3A.5. Lemma. *The following relations and functions are primitive recursive:*

$$\mathrm{ClTerm}(u) \iff u \text{ is the code of a closed term}$$
$$\iff \mathrm{Term}(u) \ \&\ (\forall i < \mathrm{lh}(u))\neg\mathrm{IndVar}((u)_i)$$

$\mathrm{CompTerm}(u) \iff u$ is the code of a term $\mathsf{p}_i^n(A_1, \dots, A_n)$
$\iff \mathrm{Term}(u)$ & $\mathrm{FunVar}(\mathrm{first}(u))$

$\mathrm{SubTerm}(u, v) \iff v$ is the code of a subterm of the term with code u
$\iff \mathrm{Term}(u)$ & $\mathrm{Term}(v)$ & $(\exists i, j \le \mathrm{lh}(u))[i < j \ \&\ v = \mathrm{seg}(u, i, j)]$

$\mathrm{PrTerm}(u) \iff u$ is the code of a term $\mathsf{p}_i^n(\mathsf{v}^1, \dots, \mathsf{v}^n)$
$\iff \mathrm{CompTerm}(u)$ & $(\forall v < u)[\mathrm{SubTerm}(u, v) \Longrightarrow \mathrm{IndVar}(v)]$

$\mathrm{Eq}(e) \iff e$ is the code of a recursive equation
$\iff e = \langle \mathrm{first}(e), \mathrm{last}(e) \rangle$ & $\mathrm{PrTerm}(\mathrm{first}(e))$ & $\mathrm{Term}(\mathrm{last}(e))$
& $(\forall i < \mathrm{lh}(\mathrm{last}(e)))\Big(\mathrm{IndVar}((\mathrm{last}(e))_i)$
$\Longrightarrow (\exists j < \mathrm{lh}(\mathrm{first}(e)))[(\mathrm{last}(e))_i = (\mathrm{first}(e))_j]\Big)$

$\mathrm{Prog}(e) \iff e$ is the code of a program
$\iff \mathrm{Seq}(e)$ & $\mathrm{lh}(e) > 0$ & $(\forall i < \mathrm{lh}(e))[\mathrm{Eq}((e)_i)]$
& $(\forall i < \mathrm{lh}(e))(\forall j < \mathrm{lh}((e)_{i,1}))$
$[\mathrm{FunVar}((e)_{i,1,j}) \Longrightarrow (\exists k < \mathrm{lh}(e))[(e)_{i,1,j} = (e)_{k,0,0}]]$

$\mathrm{State}(s) \iff s$ is the code of a state
$\iff s = \langle (s)_0, (s)_1 \rangle$
& $(\forall i < \mathrm{lh}((s)_0))\Big(\mathrm{FunSymb}((s)_{0,i}) \vee (s)_{0,i} = [?]_1 \vee \mathrm{ClTerm}((s)_{0,i})\Big)$
& $(\forall j < \mathrm{lh}((s)_1))\mathrm{IndConst}((s)_{1,j})$

$\mathrm{TermState}(s) \iff s$ is the code of a terminal state
$\iff \mathrm{State}(s)$ & $\mathrm{lh}((s)_0) = 0$ & $\mathrm{lh}((s)_1) = 1$

$\mathrm{Rep}(u, j, x) =_{\mathrm{df}}$ the result of replacing the variable
with code $j = [\mathsf{v}_i]$ by the constant x in the term with code u

$\mathrm{FullRep}(u, v, w) =_{\mathrm{df}}$ the result of replacing
the variables with codes $(v)_0, \dots, \mathrm{last}(v)$
by the constants $(w)_0, \dots, \mathrm{last}(w)$ in the term with code u

$\mathrm{TrPrTerm}(u) =_{\mathrm{df}} \langle [\mathsf{v}_{i_1}]_1, \dots, [\mathsf{v}_{i_n}]_1 \rangle \quad (\text{if } u = [\mathsf{p}^n(\mathsf{v}_{i_1}, \dots, \mathsf{v}_{i_n})]_2)$

$$\begin{aligned}\text{Transition}(e,s,s') \iff{}& \text{Prog}(e)\ \&\ \text{State}(s)\ \&\ \text{State}(s')\\ &\&\ s \to s' \text{ in } \mathcal{T}(E) \text{ for } e = [E]_4\end{aligned}$$

$$\begin{aligned}\text{Comp}(e,y) \iff{}& y \text{ is the code of a terminal computation}\\ &\text{in } \mathcal{T}(E) \text{ for } e = [E]_4\\ \iff{}& \text{Prog}(e)\ \&\ \text{Seq}(y)\ \&\ \text{lh}(y) > 1\\ &\&\ (\forall i < \text{lh}(y) \dot{-} 1)\text{Transition}(e,(y)_i,(y)_{i+1})\\ &\&\ \text{TermState}(\text{last}(y))\end{aligned}$$

$$\begin{aligned}T_n(e,\vec{x},y) \iff{}& \text{Prog}(e)\ \&\ \text{Comp}(e,y)\\ &\&\ \text{first}(\text{first}(y)) = \langle \text{first}(\text{first}(\text{first}(e)))\rangle\\ &\&\ \text{last}(\text{first}(y)) = [x_1x_2\cdots x_n]_2\end{aligned}$$

Proof. The primitive recursiveness of all the relations and functions in this list except for Transition(e,s,s') is quite simple, or at the least, routine. The computation for Transition(e,s,s') requires some work and we leave it for a problem, x3A.5*. ⊣

The three Lemmas 3A.3 – 3A.5 together complete the proof of Part (1) of Theorem 3A.1.

Proof of Part (2) of Theorem 3A.1. To prove the S^m_n-Theorem, we must compute from the code e of any program E that computes the partial function

$$g(y_1,\dots,y_m,x_1,\dots,x_n) = \varphi_e(\vec{y},\vec{x})$$

and given numbers $\vec{z} = z_1,\dots,z_m$, the code $S^m_n(e,\vec{z})$ of some other program $E_{\vec{z}}$ which computes the partial function

$$f_{\vec{z}}(\vec{x}) = g(\vec{z},\vec{x}).$$

Suppose the first equation of E is

$$p_0(\mathsf{v}_1,\dots,\mathsf{v}_m,\mathsf{v}_{m+1},\dots,\mathsf{v}_{m+n}) = E_0,$$

where we have assumed specific variables in it to simplify the computation (which would otherwise be even messier). So p_0 is the main function variable of E and

$$\overline{p}_0(\vec{y},\vec{x}) = g(\vec{y},\vec{x}).$$

Suppose q is a function variable of arity n which does not occur at all in the program E, and consider the formal equation

$$q(\mathsf{v}_{m+1},\dots,\mathsf{v}_{m+n}) = p_0(Z_1,\dots,Z_m,\mathsf{v}_{m+1},\dots,\mathsf{v}_{m+n}), \tag{72}$$

where each $Z_i \equiv S^{z_i}(0)$ is a closed, pure term which denotes the number z_i—the *numeral* of z_i, as it is sometimes called. It is quite obvious that the program

$$E_{\vec{z}} \equiv (72) \text{ followed by } E$$

computes $F_{\vec{z}}$ with main symbol q, and that we can compute its code primitive recursively from the code of E, the number i such that $q \equiv \mathsf{p}_i^n$, and the numbers $\vec{z}$;

$$\begin{aligned} &S_n^m(e, \vec{z}) \\ &\quad = \langle [\mathsf{p}_i^n(\mathsf{v}_{m+1}, \dots, \mathsf{v}_{m+n}) = p_0(Z_1, \dots, Z_m, \mathsf{v}_{m+1}, \dots, \mathsf{v}_{m+n})]_3 \rangle * e. \end{aligned}$$

So the problem of defining $S_n^m(e, \vec{z})$ comes down to finding some number i such that the function variable p_i^n does not occur in E at all, and by (25), we can just take $i = e$. Finally, we let

$$f^*(e) = [\mathsf{p}_e^n(\mathsf{v}_{m+1}, \dots, \mathsf{v}_{m+n}) = p_0(Z_1, \dots, Z_m, \mathsf{v}_{m+1}, \dots, \mathsf{v}_{m+n})]_3,$$

we check (routinely) that $f^*(e)$ is primitive recursive, and we set

$$S_n^m(e, \vec{z}) = \begin{cases} \langle f^*(e) \rangle * e, & \text{if } \mathrm{Prog}(e), \\ \langle 0, e, \vec{z} \rangle & \text{otherwise.} \end{cases}$$

The definition by cases exploits the fact that no sequence of the form $\langle 0, e, \vec{z} \rangle$ is the code of a program, which insures that $S_n^m(e, \vec{z})$ is one-to-one on all inputs. ⊣

The S_n^m-Theorem is a more-or-less obvious consequence of the fact that our codings are so-to-speak "primitive recursive", and the proof of it was messy, precisely because it required us digging into the details of these codings. It is a very useful result, as we will see, partly because we will be able to avoid a great many coding computations by appealing to it.

We confine ourselves here to just three consequence of Theorem 64 which illustrate some of its uses.

3A.6. COROLARY. *A partial function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ is recursive if and only if it is μ-recursive.*

PROOF. The μ-recursive partial functions are recursive by Corollary 2D.1, and the Normal Form Theorem obviously implies that every recursive partial function is μ-recursive. ⊣

It should be emphasized that the normal form (64) rarely—if ever—gives an efficient algorithm for the computation of a function, which in most cases is specified easily (and more naturally) by general recursion. The Ackermann function is a good example of this: its recursive definition is very simple, while a direct proof that it is μ-recursive and the derivation of a normal form for it are not quite that simple—try it.

The Normal Form Theorem is the main tool for establishing *negative results*, that certain interesting relations are *not recursive*. The basic model for such results is the following theorem of Turing:

3A.7. Corolary (Turing). *The* ***halting relation***

$$H(e,x) \iff \varphi_e(x)\downarrow \tag{73}$$

is not recursive.

Proof. Notice that directly from its definition,

$$\begin{aligned} H(e,x) &\iff (\exists y)T_1(e,x,y) \\ &\iff \mathrm{Prog}(e)\ \&\ \text{the recursive machine with code } e \\ &\qquad\qquad\qquad\qquad \text{terminates on input } x. \end{aligned}$$

If it were a recursive relation, then the total function

$$f(x) = \begin{cases} \varphi_x(x)+1, & \text{if } H(x,x) \\ 0, & \text{otherwise} \end{cases}$$

would be recursive; but then, for some e and all x we would have

$$f(x) = \varphi_e(x),$$

which is absurd for $x = e$. ⊣

• $\varphi_e(x) = \varphi_x(x) + 1$ need not be true for all x, only for those x's for which $\varphi_x(x)\downarrow$; but $\varphi_e(e)\downarrow$, because $\varphi_e(x)$ is total, which gives the contradiction on the next line.

Our third Corollary is an application of the S^m_n-Theorem:

3A.8. Corolary. *Primitive recursion is primitive recursive in the codes; i.e., for each n, there exists a primitive recursive function $u(e,m) = u^n(e,m)$ such that if the partial function f is defined by the primitive recursion*

$$\begin{aligned} f(0,\vec{x}) &= \varphi^n_e(\vec{x}) \\ f(y+1,\vec{x}) &= \varphi^{n+2}_m(f(y,\vec{x}),y,\vec{x}), \end{aligned}$$

then

$$f(y,\vec{x}) = \varphi^{n+1}_{u(e,m)}(y,\vec{x}).$$

Proof. The partial function

$$\begin{aligned} g(0,e,m,\vec{x}) &= \varphi^n(e,\vec{x}) \\ g(y+1,e,m,\vec{x}) &= \varphi^{n+2}(m,g(y,e,m,\vec{x}),y,\vec{x}) \end{aligned}$$

is recursive, because $\mathcal{R}$ is closed under primitive recursion, so

$$h(e,m,y,\vec{x}) = g(y,e,m,\vec{x}),$$

is also recursive and so it has some code $\widehat{h}$. It follows that

$$g(y,e,m,\vec{x}) = h(e,m,y,\vec{x}) = \varphi^{n+3}_{\widehat{h}}(e,m,y,\vec{x}) = \varphi^{n+1}_{S^2_{n+1}(\widehat{h},e,m)}(y,\vec{x}),$$

and we can set

$$u(e,m) = S^2_{n+1}(\widehat{h},e,m).$$ ⊣

Problems for Section 3A

x3A.1. Prove Lemma 3A.2.

x3A.2. Prove that every recursive partial function $f(\vec{x})$ has infinitely many codes, i.e., there exist infinitely many numbers e such that $f = \varphi^n_e$.

x3A.3. Prove that if $T_n(e, \vec{x}, y)$ is defined by (67) and $U(y)$ is defined by (70), then (68) holds.

x3A.4. Give a precise definition of the relation $R_4(d, j)$ in the proof of Lemma 3A.4.

x3A.5*. Prove that the relation Transition(e, s, s') is primitive recursive. (This computation has many details and it is not feasible to record them all. What is required in this Problem is to explain the architecture of the proof, and to work out some of the more interesting cases.)

x3A.6. Prove that some primitive recursive function $u(n)$ gives for each n a code of the Ackermann section $A_n(x)$.

x3A.7. Prove that the composition of recursive partial functions is primitive recursive in the codes, in the following sense: for some primitive recursive function $u(z, e, m)$,

$$\varphi^n_{u(z,e,m)}(\vec{x}) = \varphi^2_z(\varphi^n_e(\vec{x}), \varphi^n_m(\vec{x})).$$

x3A.8*. Prove that there is a recursive partial function $f(x)$ which does not have a total recursive extension, i.e., there is no total recursive function g such that $f \sqsubseteq g$.

3B. The Church-Turing Thesis

Our main aim in this section is to explain and (at least partly) justify the following fundamental principle. It was formulated by the American Alonzo Church and the British Alan Turing in 1936, independently and in different forms.

3B.1. **Church-Turing Thesis** CT. *A function $f : \mathbb{N}^n \to \mathbb{N}$ is computable if and only if it is recursive.*

The Church-Turing Thesis identifies our intuitive understanding of what it means for a function $f : \mathbb{N}^n \to \mathbb{N}$ to be *computable by some algorithm* with the precise claim that f is recursive. It is usually claimed for partial functions $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ also, but most of its (many and important) applications depend on the simpler formulation for total functions.

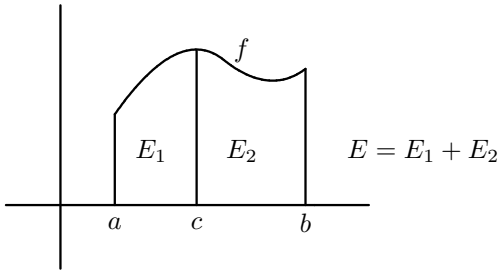


Figure 4. Area as integral.

The implication

$$f \text{ recursive} \implies f \text{ computable}$$

is the *trivial direction* of the Thesis, and it is generally considered obvious "by definition": today, if not in 1936, it is easy to view the recursive machines defined in 2B.5 as "ideal versions" of electronic computers, or even as real computers with access to "unlimited memory", and they certainly compute the values of their least fixed points. The significance of the Thesis lies in its *essential direction*,

$$f \text{ computable} \implies f \text{ recursive}$$

typically invoked in the contrapositive:

$$\text{if } f \text{ is not recursive, then no algorithm computes } f.$$

This allows to prove rigorously that a specific function f is *absolutely non-computable*—not computable by any algorithm—by verifying that f is not recursive.

The Church-Turing Thesis cannot be proved rigorously, as it identifies the intuitive notion of *computability* with the rigorous (mathematical) notion of *recursiveness*. In other words, it cannot be a *theorem*, so that within mathematics it has the status of a *definition*. To understand its meaning, it is useful to examine the role that definitions play in mathematics—*how they are justified* and *what they are used for*.

For a classical example, consider the *area* of a *simple region* E bounded by the x-axis, the vertical lines $x = a$ and $x = b$ and the graph of a positive, continuous function f. It is defined in calculus by an integral,

$$\text{Area}(E) = \int_a^b f(x)dx. \tag{74}$$

This is not arbitrary, of course, or it would not be useful. The integral formula (74) gives the correct value in important applications, e.g., it predicts

that the area of a circle with radius r is πr^2, which can then verified by measurements.

In other words, the first requirement for a mathematical definition of some intuitive notion is that *it agrees with the known examples*, those which make the notion interesting. For the Church-Turing Thesis, this comes down to arguing that

(75) *if a known algorithm computes $f : \mathbb{N}^n \to \mathbb{N}$, then f is recursive.*

This is not in dispute today: apart from the long list of (primitive) recursive functions in section 1B and the strong closure properties of the class of recursive partial functions that we have proved, there is also the experience of more than seventy years of research which have yielded no "counterexample"—no function which is considered (generally by the mathematical community) to be computable but which is not recursive.

Can we be sure that no counterexample of the Church-Turing Thesis will be found in the future? Consider the classical solution of the corresponding problem for the notion of area, which is based on the following three basic intuitions for simple regions as in Figure 4:

(1) The area of a rectangle with sides α and β is the product $\alpha\beta$.
(2) If $E_1 \subseteq E$, i.e., the simple region E_1 is a part of another E, then $\text{Area}(E_1) \leq \text{Area}(E)$.
(3) If the simple region E is the union of two simple regions E_1 and E_2 as in Figure 4, then $\text{Area}(E) = \text{Area}(E_1) + \text{Area}(E_2)$.

If we assume these as axioms, then equation (74) is a theorem: it can be shown that it gives *the only way to assign a real number* $\text{Area}(E)$ *to every simple region E so that* (a) – (c) *hold.*[4]

Turing made an extensive analysis of what it means for a function to be "computable" which led him to propose his version of CT. It was based on the simple and obvious intuition[5] that

(76) computability = mechanical computability.

---

[4]This is, in effect, the standard result that *the Riemann integral $\int_a^b f(x)dx$ exists for every continuous function f.*

[5]It may be a "simple" and "obvious" intuition today, when high school students are regularly using computers and the words "computation" and "mechanical computation" are viewed as synonyms. In 1936, however, electronic computers, programs and operating systems did not exist, and the mathematical tradition associated the notion of "computable function" with mathematical *algorithms* typically expressed by recursive definitions. The archetypical examples of computable functions were the numerical operations computed by the so-called "school algorithms" (for multiplication and division in the decimal system), and the greatest common divisor, computed by the Euclidean algorithm x1C.10.

After that, Turing defined rigorously a specific class of abstract (and very simple) machines which today bear his name, the *Turing machines*; he argued convincingly that *the computations of any "constructible" machine can be "simulated" by some Turing machine*; and, finally, he conjectured that for every function f,

$$f \text{ is computable} \iff f \text{ is computable by a Turing machine.}$$

To complete the story, Church had already proposed a little earlier the claim that for every f,

$$f \text{ is computable} \iff f \text{ is } \lambda\text{-definable,}$$

where "λ-definable" means "computable by some term" of the formal language of the λ-calculus which he had introduced. The equivalences

$$f \text{ is Turing computable} \iff f \text{ is } \lambda\text{-definable} \iff f \text{ is recursive}$$

were proved almost immediately afterwards (by Turing and Kleene), and they closed the circle which led to the formulation of the Church-Turing Thesis in the form 3B.1.

Whether the analysis of "mechanical computability" given by Turing constitutes a complete justification of the Church-Turing Thesis (just as the analysis we outlined justifies the correctness of the definition of area for simple regions) is debatable—and sometimes debated hotly. In any case, we will consider it briefly in the next Section but we will not elaborate on it here. We will show instead that the mechanical computability of a partial function f implies its recursiveness if the machine that does the computing satisfies some very weak "effectivity conditions" which are satisfied by all ideal computers (as we understand these today) and by all "models of computation" that have been introduced.

3B.2. A **recursive coding** of an abstract machine (as in 2B.4)

$$\mathcal{M} = (S, T, \to, \text{input}, \text{output})$$

with input set $\mathbb{N}^n$ and output set $\mathbb{N}$ is any coding

$$c : S \rightarrowtail \mathbb{N}$$

of the states of $\mathcal{M}$ in the natural numbers such that the following functions, relations and sets are recursive:

$$\begin{aligned} S_c &= c[S] = \{x \mid (\exists s \in S)[c(s) = x]\} \\ T_c &= c[T] = \{x \mid (\exists s \in T)[c(s) = x]\} \\ x \to_c x' &\iff x, x' \in S_c \ \&\ c^{-1}(x) \to c^{-1}(x') \\ &\iff (\exists s, s' \in S)[c(s) = x \ \&\ c(s') = x' \ \&\ s \to s'] \end{aligned}$$

$$\text{input}_c(\vec{x}) = c(\text{input}(\vec{x})), \qquad \text{output}_c(x) = \begin{cases} \text{output}(c^{-1}(x)), & \text{if } x \in T_c, \\ 0, & \text{otherwise.} \end{cases}$$

3B.3. Theorem. *Every partial function computed by an abstract machine which admits a recursive coding is recursive.*

Proof. The proof is a small part of the proof of the Normal Form Theorem 3A.1, without the computations which are now given by the hypothesis. We set

$$\begin{aligned} C(\vec{x},y) &\iff y \text{ is the code of a computation on input } \vec{x} \\ &\iff \mathrm{Seq}(y) \ \&\ (y)_0 = \mathrm{input}_c(\vec{x}) \\ &\qquad \&\ (\forall i < \mathrm{lh}(y))[i+1 < \mathrm{lh}(y) \Longrightarrow (y)_i \to_c (y)_{i+1}] \\ &\qquad \&\ (y)_{\mathrm{lh}(y) \dot{-} 1} \in T_c. \end{aligned}$$

The relation $C(\vec{x},y)$ is recursive by the hypothesis and the closure properties of recursive relations, and if the given machine computes the partial function f, then

$$f(\vec{x}) = \mathrm{output}_c\Big(\Big(\mu y C(\vec{x},y)\Big)_{\mathrm{lh}(y) \dot{-} 1}\Big),$$

so that $f(\vec{x})$ is recursive. ⊣

The idea now is that if a machine $\mathcal{M}$ (with input set $\mathbb{N}^n$ and output set $\mathbb{N}$) can be "constructed" in some precise sense, then the states of $\mathcal{M}$ must be finite objects, which can be coded simply in the natural numbers (as in the proof of 3A.1), so that the basic relations are recursive, in other words, the proposition that

every machine which can be built admits a recursive coding;

and the basic observation is that the Church-Turing Thesis follows rigorously from this claim and Turing's basic intuition (76).

3C. Symbolic computation and undecidability

Sometimes we appeal to the Church-Turing Thesis to avoid giving a rigorous proof, i.e., we claim that some partial function is recursive by giving an intuitive description of some algorithm which computes it. Such "sinful" (lazy) appeals can certainly be avoided with some additional work. The Church-Turing Thesis is used in an essential way in proofs of *negative results*, *non-computability* of functions or *undecidability* of relations, usually relations on the set of words from some finite alphabet.

3C.1. **Undecidability**. Let $\Sigma = \{a_0, \dots, a_r\}$ be a finite (non-empty) alphabet, and let Σ^* the set of all words (finite sequences) from Σ, including the empty word Λ. We set $[a_i] = i$ and code Σ^* in $\mathbb{N}$ in the simplest way:

$$[s_0 \dots s_{k-1}] = \langle [s_0], \dots, [s_{k-1}] \rangle \quad (s_0 \dots s_{k-1} \in \Sigma).$$

An n-place relation (or "problem") P on Σ^* is *decidable* or *solvable* if there exists a recursive function $\chi(x_1, \dots, x_n)$ such that

$$P(u_1, \dots, u_n) \iff \chi([u_1], \dots, [u_n]) = 1 \quad (u_1, \dots, u_n \in \Sigma^*),$$

otherwise P is *undecidable* or *unsolvable*.

The basic tool for undecidability proofs is a characterization of the decidable relations on Σ^* in terms of some *model of symbolic computation* directly, i.e., without reference to codings. In Section 3D we will review briefly the classical (and historically first) such model, the famous Turing *machines*. Here we introduce a second classical example due to Post, who also proved the first undecidability result in pure mathematics, outside of mathematical logic and computability theory.

3C.2. A **rewriting system** (or semi-Thue system) is a structure

$$R = (\Sigma, \sigma_0 \to \tau_0, \dots, \sigma_k \to \tau_k) = (\{a_0, \dots, a_n\}, \sigma_0 \to \tau_0, \dots, \sigma_k \to \tau_k),$$

where the *alphabet* Σ is a finite set and in the *transition rules* $\sigma_i \to \tau_i$ the sequences σ_i, τ_i are (possibly empty) words from Σ, i.e., members of Σ^*. The system R defines the following *basic transition relation* on Σ^*:

$$\alpha \to_R \beta \iff (\exists \alpha_1, \alpha_2, \sigma, \tau \in \Sigma^*)[\sigma \to \tau \ \&\ \alpha = \alpha_1 \sigma \alpha_2 \ \&\ \beta = \alpha_1 \tau \alpha_2],$$

i.e., $\alpha \to_R \beta$ if β is derived from α by the replacement of some part σ of α by some word τ, so that $\sigma \to \tau$ is a rule. For example, in the rewriting system

$$R = (\{a, b\}, a \to aa, a \to bb),$$

we have the basic rewritings

$$a \to_R aa \to_R aaa \to_R abba.$$

The (complete) *transition relation* of R is the "transitive closure" of $\to_R$,

$$\begin{aligned} \alpha \to_R^* \beta &\iff \alpha \to_R \alpha_1 \to_R \cdots \to_R \alpha_{n-1} \to_R \beta \\ &\iff (\exists \alpha_0, \dots, \alpha_n)[\alpha_0 = \alpha \\ &\qquad\qquad \& (\forall i < n)[\alpha_i \to_R \alpha_{i+1} \ \&\ \alpha_n = \beta], \end{aligned}$$

so that in the example $a \to_R^* abba$, but $a \not\to_R^* bbb$ (Problem x3C.1). The word α is **terminal** in R if there is no β such that $\alpha \to_R \beta$,

$$T_R = \{\alpha \mid (\forall \beta)[\alpha \not\to_R \beta],$$

like any $\alpha \in \{b\}^*$ in the example. Note that in these general rewriting systems, we do not separate into "terminal" and "stuck" words, i.e., we call terminal all stuck words.

Finally, for $S \subseteq \Sigma^*$, the system R is **deterministic on** S, if

$$[\alpha \in S \ \&\ \alpha \to_R \beta \ \&\ \alpha \to_R \beta'] \Longrightarrow \beta = \beta' \in S,$$

so that the example is not deterministic on the complete $\{a, b\}^*$, but it is deterministic on $\{b\}^*$.

The recursive transition systems $\mathcal{T}(E, \mathbf{M})$ are, basically, rewriting systems, except that we distinguish "terminal" from "stuck" words and (more significantly) the alphabet Σ is infinite, as we have to include all constants, i.e., all natural numbers in the case $\mathbf{M} = \mathbf{N}_0$ which concerns us. The next, technical but basic result gets around this obstacle by setting

$$\mathbf{x} = \$\underbrace{11\cdots1}_{x}\$ \quad (x \in \mathbb{N}), \tag{77}$$

i.e., coding numbers with strings of 1's.

3C.3. Theorem. *Every recursive, partial function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ is computed by a rewriting system R, in some* (finite) *set of symbols Σ which contains the special symbols* $\mathsf{f}, 1, \$, :$ *in the following sense:*

$$f(x_1, \dots, x_n) = w \iff \mathsf{f} : \mathbf{x}_1\mathbf{x}_2\cdots\mathbf{x}_n \to_R^* \ : \mathbf{w}.$$

Moreover, R is deterministic on the set

$$S = \{\alpha : \beta \mid \text{the symbol ':' does not occur in the words } \alpha, \beta\}. \tag{78}$$

We omit the proof which involves a good deal of "programming" in rewriting systems.

3C.4. **The word problem for semigroups**. For each rewriting system

$$R = (\Sigma, \sigma_0 \to \tau_0, \dots, \sigma_k \to \tau_k), \tag{79}$$

let $\sim_R$ be the smallest equivalence relation on Σ^* such that

1. $\sigma_i \sim_R \tau_i$, for $i \leq k$,
2. $\sigma \sim_R \tau \Longrightarrow \alpha\sigma\beta \sim_R \alpha\tau\beta$, for all $\alpha, \beta \in \Sigma^*$,

and for every $\alpha \in \Sigma^*$, let

$$[\![\alpha]\!] = \{\beta \in \Sigma^* \mid \beta \sim_R \alpha\}$$

be the *equivalence class* of u. The set of the equivalence classes

$$[\![R]\!] = \{[\![\alpha]\!] \mid \alpha \in \Sigma^*\}$$

is the *semigroup* (generated by Σ and presented by R), and the relation $\sim_R$ is called the *word problem* for this semigroup. The terms are, obviously, from algebra and they can be justified, but we will not do this here.

3C.5. Theorem (Emil Post). *There is a rewriting system R such that the relation $\alpha \to_R^* \beta$ is not decidable, and the word problem $\alpha \sim_R \beta$ for the semigroup $[\![R]\!]$ is unsolvable.*

PROOF. Let $R_f = (\Sigma, \sigma_0 \to \tau_0, \dots, \sigma_k \to \tau_k)$ be a rewriting system which computes the recursive partial function

$$f(x) = 0 \cdot \mu y T_1(x, x, y)$$

by Theorem 3C.3, so that

$$f(x)\downarrow \iff \mathsf{f} : \mathbf{x} \to_R^* \ : \mathbf{0};$$

it follows that the relation $\alpha \to_R^* \beta$ is not decidable, because if it were, then the domain of definition $\{x \mid f(x)\downarrow\}$ would be recursive, which it is not, 3A.7.

To infer that the relation $\sim_R$ is not decidable either, we observe first that

$$\begin{aligned}\alpha \sim_R \beta \iff (\exists \gamma_0, \dots, \gamma_n)[&\alpha = \gamma_0 \ \&\ \gamma_n = \beta \\ &\&\ (\forall i < n)[\gamma_i = \gamma_{i+1} \vee \gamma_i \to_R \gamma_{i+1} \vee \gamma_{i+1} \to_R \gamma_i]],\end{aligned}$$

easily in the direction $\Rightarrow$ by the definition of $\sim_R$ (since the relation on the right is an equivalence relation), and by a trivial induction on n in the direction $\Leftarrow$. It follows that it is enough to show that (with S as in (78))

$$\begin{aligned}(80)\quad (\forall i < n)[\gamma_i = \gamma_{i+1} \vee \gamma_i \to_R \gamma_{i+1} \vee \gamma_{i+1} \to_R \gamma_i] \\ \&\ \gamma_0 \in S \ \&\ \gamma_n = \ : \mathbf{0} \\ \Longrightarrow \gamma_0 = \ : \mathbf{0} \vee \gamma_0 \to_R^* \ : \mathbf{0},\end{aligned}$$

which implies that for $\alpha \in S$,

$$\alpha \sim_R \ : \mathbf{0} \iff \alpha = \ : \mathbf{0} \vee \alpha \to_R^* \ : \mathbf{0},$$

so that

$$\mathsf{f} : \mathbf{x} \to_R^* \mathbf{0} \iff \mathsf{f} : \mathbf{x} \sim_R \mathbf{0}$$

and excludes the decidability of $\sim_R$ as before. Finally, we show (80) by induction on n and with trivial basis, since for $n = 0$ the condition (80) says that $w_0 = \ : \mathbf{0} \Longrightarrow w_0 = \ : \mathbf{0}$. The induction step is also trivial if for some i, $w_i = w_{i+1}$, or if, for every $i < n$, $w_i \to_R w_{i+1}$. On the other hand, the state $\ : \mathbf{0}$ is terminal, so $w_{i+1} \to_R w_i$ cannot hold for every i, and so there remains the case that for some *largest* i,

$$w_{i+1} \to_R w_i,$$

and therefore, also,

$$w_{i+1} \to_R w_{i+2},$$

and from these two and the determinism of R on S it follows that $w_i = w_{i+2}$, which with the induction hypothesis complete the proof, because we can simply remove w_{i+1} from the given sequence. $\dashv$

From the numerous mathematical problems which have been proved unsolvable we mention here only two.

3C.6. **The word problem for groups**. For every finite alphabet $\Sigma = \{a_0, \dots, a_n\}$, let

$$\Sigma_g = \Sigma \cup \{a_0^{-1}, \dots, a_n^{-1}\}$$

where $a_0^{-1}, \dots, a_n^{-1}$ are new symbols; and for every rewriting system

$$R = (\Sigma_g, \sigma_0 \to \tau_0), \dots, \sigma_k \to \tau_k))$$

in Σ_g^*, let $\simeq_R$ be the smallest equivalence relation on Σ_g^* such that

1. $\sigma_i \simeq_R \tau_i$, for $i \leq k$,
2. $a_i a_i^{-1} \simeq_R a_i^{-1} a_i \simeq_E \Lambda$, where Λ the empty word,
3. $\sigma \simeq_R \tau \Longrightarrow \alpha\sigma\beta \sim_R \alpha\tau\beta$, for all $\alpha, \beta \in \Sigma^*$,

and for every $\alpha \in \Sigma_g^*$, let

$$[\![\alpha]\!]_g = \{\beta \in \Sigma_g^* \mid \beta \simeq_R \alpha\}$$

the *equivalence class* of α. The set of the equivalence classes

$$[\![R]\!]_g = \{[\![\alpha]\!]_g \mid \alpha \in \Sigma_g^*\}$$

is the *group* generated by R, and the equivalence relation $\simeq_R$ is called the *word problem* for this group. As for the semigroups, the terms obviously originate from algebra and can be justified, but we will not do this here.

3C.7. THEOREM (S. Novikov, W. Boone). *There exists a rewriting system R such that the word problem for the group $[\![R]\!]_g$ is unsolvable.*

This difficult result has important corollaries for group theory and algebraic topology.

3C.8. THEOREM (**Hilbert's 10th Problem**). *The problem whether a given Diophantine polynomial*

$$P(x_1, \dots, x_n) = \sum_{k_1 + \dots + k_n \leq d} a_{k_1, \dots, k_n} x_1^{k_1} \cdots x_n^{k_n}$$

in n variables with coefficients in $\mathbb{Z} = \{\dots, -1, 0, 1, \dots\}$ has integer roots is unsolvable.

This theorem was proved by Yuri Matijasevich in 1970 following a great deal of work by Hilary Putnam, Julia Robinson and Martin Davis and 70 years after the question was asked. Perhaps more than any other result, it helped establish recursion theory as a subject with significant applications in pure mathematics.

Problems for Section 3C

x3C.1. For which words $x_1 x_2 \cdots x_n$ does the relation $a \to_R^* x_1 x_2 \cdots x_n$ hold, for the rewriting system $R = (\{a, b\}, \{a \to aa, a \to bb\}$?

3D. Turing machines

We define here Turing machines and give a brief account (without proof) of the main fact about them.

3D.1. Definition. A **Turing machine** is a triple $\mathcal{M} = (\Sigma, S^i, T)$ where:

(1) Σ is a finite set of *symbols*, which includes the special *blank symbol* $\sqcup$.
(2) S^i is a non-empty, finite set, the set of *internal states* of $\mathcal{M}$. We assume that $\Sigma \cap S^i = \emptyset$.
(3) T is a finite set of quintuples of the form

$$Q \xrightarrow{x\ x'\ m} Q' \tag{81}$$

where Q and Q' are internal states, x and x' are symbols, and the *move* m is 0, $+1$ or -1. This is the *set of transitions* or *table* of $\mathcal{M}$.

A machine $\mathcal{M}$ is **deterministic** if for every internal state Q and every symbol x, there is at most one transition $Q \xrightarrow{x\ x'\ m} Q'$ in the table of $\mathcal{M}$ which starts with (Q, x).

In the picture that Turing sketches, at each step of its computation the machine is in a particular internal state Q and is "looking" at one "cell" of an infinite (in both directions) "tape" with finitely many non-empty cells. We picture this (complete) *state* of $\mathcal{M}$ by the *doubly infinite word*

$$\alpha\, Q\, \beta \equiv \cdots \alpha(n) \cdots \alpha(0)\, Q\, \beta(0) \cdots \beta(n) \cdots$$

in the alphabet $\Sigma \cup S^i$, i.e., Q is an internal state and $\alpha(n), \beta(n) \in \Sigma$ for every n. The *visible symbol* of $\alpha\, Q\, \beta$ is $\beta(0)$, and the pair $(Q, \beta(0))$ determines completely how $\mathcal{M}$ will transform the state.

For example, if the internal state is Q and the tape is empty, then

$$\text{the state is } \sqcup^{-\infty} Q \sqcup^{\infty} = \ldots \sqcup\sqcup Q \sqcup\sqcup \cdots \text{ and the visible symbol is } \sqcup.$$

Briefly, all the machine can do is to change internal state, change the visible symbol and move no more than one position to the left or to the right. The possibilities are determined by the entries in the table of $\mathcal{M}$ and they are illustrated compactly in Table 2, in the second column with Turing's notation, and in the third as a transition system on the set of states

$$\{\alpha, Q, \beta \mid \alpha, \beta \in \Sigma^*, Q \in S\};$$

this is a bit different, because we need to deal with the states

$$\alpha Q \Lambda \qquad \Lambda Q \beta \qquad \Lambda Q \Lambda$$

where the empty word on the left represents $\sqcup^{-\infty}$ and on the right $\sqcup^{\infty}$. They can be described as follows, assuming that

$$\text{the state is } \alpha\, Q\, \beta \text{ and } x \equiv \beta(0).$$

	$\cdots \quad t \quad x \quad y \quad \cdots$ $\uparrow$ Q	$\alpha, \beta \in \Sigma^*$
$Q \xrightarrow{x\ x'\ +1} Q'$	$\cdots \quad t \quad x' \quad y \quad \cdots$ $\uparrow$ Q'	$\alpha Q x \beta \to \alpha x' Q' \beta$ $\alpha Q \Lambda \to \alpha x' Q' \Lambda$
$Q \xrightarrow{x\ x'\ 0} Q'$	$\cdots \quad t \quad x' \quad y \quad \cdots$ $\uparrow$ Q'	$\alpha Q x \beta \to \alpha Q' x' \beta$ $\alpha Q \Lambda \to \alpha Q' x' \Lambda$
$Q \xrightarrow{x\ x'\ -1} Q'$	$\cdots \quad t \quad x' \quad y \quad \cdots$ $\uparrow$ Q'	$\alpha y Q x \beta \to \alpha Q' y x' \beta$ $\alpha y Q \Lambda \to \alpha Q' y x' \Lambda$ $\Lambda Q x \beta \to \Lambda Q' \sqcup x' \beta$

Table 2. The transitions of a non-deterministic Turing machine.

(0) If there is no basic transition in the table which starts with Q and x, then the complete situation is declared *terminal* and the computation stops.

(1) If the table has a transition

$$Q \xrightarrow{x\ x'\ 0} Q'$$

with move $m = 0$, then the machine "chooses" such a transition, changes x to x' and changes its internal state from Q to Q', i.e.,

$$\alpha\, Q\, x \beta' \to \alpha\, Q'\, x'\, \beta'$$

(2) If the table has a transition

$$Q \xrightarrow{x\ x'\ +1} Q'$$

with move $m = 1$, then the machine "chooses" such a transition, changes x to x', changes its internal state from Q to Q' and moves one place to the right, i.e.,

$$\alpha\, Q\, x \beta' \to \alpha x'\, Q'\, \beta'$$

(3) If the table has a transition

$$Q \xrightarrow{x\ x'\ -1} Q'$$

with move $m = -1$, then the machine "chooses" such a transition, changes x to x', changes its internal state from Q to Q' and moves one place to the left, i.e.,

$$\alpha' y\, Q\, x \beta' \to \alpha'\, Q'\, y x' \beta'$$

With this definition, a Turing machine is a transition system by Definition 2B.2 and the various notions associated with transition systems apply to it: a *computation* of $\mathcal{M}$ is any (finite or infinite) sequence

$$s_0, s_1, \dots, s_k, \dots$$

where each s_i is a complete state and s_{i+1} is the result of an action of the machine at s_i;

$$s \to^* s' \iff (\exists \text{ computation})[s, s_1, \dots, s_{n-1}, s'];$$

and if $\mathcal{M}$ is deterministic, then there is exactly one (terminal or infinite) computation which starts with any complete state $Q \xrightarrow{x\ x'\ m} Q'$. (We do not distinguish between *stuck* and *terminal* complete states in this version of Turings's definition.)

To compute a (partial) function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ with a Turing machine we need an input and and output function, and the most usual choice for these are

$$\text{input}(\vec{x}) = \sqcup^\infty \text{START } \underbrace{11\cdots1}_{x_1+1} \sqcup \underbrace{11\cdots1}_{x_2+1} \sqcup \cdots \sqcup \underbrace{11\cdots1}_{x_n+1} \sqcup^\infty$$

$$\text{output}(\sqcup^\infty \text{END } \underbrace{11\cdots1}_{w+1} \sqcup \beta = w,$$

where START and END are specified internal states.

3D.2. Theorem. *Every recursive partial function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ can be computed by a deterministic Turing machine.*

This basic result is proved by showing that the class of Turing computable partial functions is primitively closed and also closed under minimalization, so that it contains all recursive partial functions. It is a long and rather tedious exercise in "Turing machine programming" and we will skip it.

The converse is an easy consequence of Theorem 3B.3, so

3D.3. **Theorem**. *A partial function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ is* (deterministically or non-deterministically) *Turing-computable if and only if it is recursive.*

CHAPTER 4

RECURSIVELY ENUMERABLE SETS

Recall that a relation $R(\vec{x})$ on the natural numbers is **recursive** if its characteristic function is recursive, and a set $A \subseteq \mathbb{N}$ is **recursive** if its membership relation

$$R_A(x) \iff x \in A$$

is recursive. We will study here the basic mathematical properties of the recursive (partial) functions, relations and sets, but also the *semirecursive relations* and the *recursively enumerable sets*, the simplest non-computable mathematical objects. The basic tools we will use are the robust closure of the class of recursive partial functions $\mathcal{R}$ (2C.3, 2C.4, 2D.1), the Normal Form and Enumeration Theorem 3A.1 and the 2nd Recursion Theorem 4D.1.

In appealing to Theorem 3A.1, we will often simplify the notation by omitting the superscript

$$\varphi_e(\vec{x}) = \varphi_e^n(\vec{x}) = U(\mu y T_n(e, \vec{x}, y))$$

(which can be read off the input $\vec{x} = x_1, \dots, x_n$) and using in some cases Kleene's alternative notation,

$$\{e\}(\vec{x}) = \varphi_e^n(\vec{x}) = U(\mu y T_n(e, \vec{x}, y)). \tag{82}$$

This is "typographically" convenient (fewer super- and sub- scripts), but also helps understand better some of the proofs, as it places "on the same level" the *program* e and the *data* $\vec{x}$. Finally, we will also use the notation

$$W_e = \{x \mid \varphi_e(x)\downarrow\} \tag{83}$$

for the domain of convergence of the recursive partial function with code e.

4A. Semirecursive relations

To facilitate the formulation of definitions and results in the sequel, we list here and name the most basic operations on relations.

$(\neg)$	$P(\vec{x}) \iff \neg Q(\vec{x})$		(negation)
$(\&)$	$P(\vec{x}) \iff Q(\vec{x}) \ \& \ R(\vec{x})$		(conjunction)
$(\vee)$	$P(\vec{x}) \iff Q(\vec{x}) \vee R(\vec{x})$		(disjunction)
$(\Rightarrow)$	$P(\vec{x}) \iff Q(\vec{x}) \Longrightarrow R(\vec{x})$		(implication)
$(\exists)$	$P(\vec{x}) \iff (\exists y)Q(\vec{x}, y)$		(existential quantification)
$(\exists_{\leq})$	$P(z, \vec{x}) \iff (\exists i \leq z)Q(\vec{x}, i)$		(bounded ex. quant.)
$(\forall)$	$P(\vec{x}) \iff (\forall y)Q(\vec{x}, y)$		(universal quantification)
$(\forall_{\leq})$	$P(z, \vec{x}) \iff (\forall i \leq z)Q(\vec{x}, i)$		(bounded univ. quant.)
(84)	$P(\vec{x}) \iff Q(f_1(\vec{x}), \dots, f_m(\vec{x}))$		(substitution)

For example, we have already shown that the set of primitive recursive relations is closed under all these operators (with primitive recursive $f_i(\vec{x})$) except for the (unbounded) quantifiers $\exists, \forall$, under which it is not closed by Theorem 3A.7. We have also shown (mostly in problems) the closure properties of the recursive relations:

4A.1. Proposition. *The set of recursive relations is closed under the propositional operators* $\neg$, $\&$, $\vee, \Rightarrow$, *the bounded quantifiers* $\exists_{\leq}, \forall_{\leq}$ *and substitution of* (total) *recursive functions, but it is not closed under the quantifiers* $\exists, \forall$.

4A.2. Definition. (1) A relation $P(\vec{x})$ is **semirecursive** if for some recursive partial function $f(\vec{x})$,

$$P(\vec{x}) \iff f(\vec{x})\downarrow .$$

(2) A relation $P(\vec{x})$ is Σ^0_1 if for some recursive relation $Q(\vec{x}, y)$

$$P(\vec{x}) \iff (\exists y)Q(\vec{x}, y).$$

4A.3. Proposition. *The following are equivalent, for any relation* $P(\vec{x})$:
(1) $P(\vec{x})$ *is semirecursive.*
(2) $P(\vec{x})$ *is* Σ^0_1.
(3) $P(\vec{x})$ *satisfies an equivalence*

$$P(\vec{x}) \iff (\exists y)Q(\vec{x}, y)$$

with some primitive recursive $Q(\vec{x}, y)$.

Proof. (1) $\Longrightarrow$ (3) by the normal form theorem; (3) $\Longrightarrow$ (2) trivially; and (2) $\Longrightarrow$ (1) setting

$$f(\vec{x}) = \mu y Q(\vec{x}, y),$$

so that

$$(\exists y)Q(\vec{x}, y) \iff f(\vec{x})\downarrow . \qquad \dashv$$

We have already seen the classical, simplest example of a relation which is semirecursive but not recursive, the halting relation $H(e,x)$ in Corollary 3A.7.

4A.4. Proposition (Kleene's Theorem). *A relation $P(\vec{x})$ is recursive if and only if both $P(\vec{x})$ and its negation $\neg P(\vec{x})$ are both semirecursive.*

Proof. If $P(\vec{x})$ is recursive, then

$$Q(\vec{x},y) \iff P(\vec{x}), \qquad R(\vec{x},y) \iff \neg P(\vec{x})$$

are also recursive and (trivially, by *vacuous quantification*)

$$\begin{aligned} P(\vec{x}) &\iff (\exists y)Q(\vec{x},y)\\ \neg P(\vec{x}) &\iff (\exists y)R(\vec{x},y). \end{aligned}$$

For the other direction, if

$$\begin{aligned} P(\vec{x}) &\iff (\exists y)Q(\vec{x},y)\\ \neg P(\vec{x}) &\iff (\exists y)R(\vec{x},y) \end{aligned}$$

with recursive relations Q and R, then the function

$$f(\vec{x}) = \mu y[R(\vec{x},y) \vee Q(\vec{x},y)]$$

is recursive, total, and

$$P(\vec{x}) \iff Q(\vec{x},f(\vec{x})). \qquad \dashv$$

4A.5. Proposition. *The set of semirecursive relations is closed under recursive substitutions, under the "positive" propositional operators $\&,\vee$, under the bounded quantifiers $\exists_{\leq}$, $\forall_{\leq}$, and under the existential quantifier $\exists$; it is not closed under negation $\neg$ or under the universal quantifier $\forall$.*

Proof. Closure under recursive substitutions is obvious, and the following transformations show the remaining, positive claims of the proposition:

$$\begin{aligned} (\exists y)Q(\vec{x},y) \vee (\exists y)R(\vec{x},y) &\iff (\exists u)[Q(\vec{x},u) \vee R(\vec{x},u)]\\ (\exists y)Q(\vec{x},y) \;\&\; (\exists y)R(\vec{x},y) &\iff (\exists u)[Q(\vec{x},(u)_0) \;\&\; R(\vec{x},(u)_1)]\\ (\exists z)(\exists y)Q(\vec{x},y,z) &\iff (\exists u)R(\vec{x},(u)_0,(u)_1)\\ (\exists i \leq z)(\exists y)Q(\vec{x},y,i) &\iff (\exists u)[(u)_0 \leq z \;\&\; Q(\vec{x},(u)_1,(u)_0)\\ (\forall i \leq z)(\exists y)Q(\vec{x},y,i) &\iff (\exists u)(\forall i \leq z)Q(\vec{x},(u)_i,i). \end{aligned}$$

On the other hand, the set of semirecursive relations is not closed under negation or the universal quantifier, since otherwise the basic halting relation

$$H(e,x) \iff (\exists y)T_1(e,x,y)$$

would have a semirecursive negation and so it would be recursive by 4A.4, which it is not. $\dashv$

The **graph** of a partial function $f(\vec{x})$ is the relation

$$G_f(\vec{x}, w) \iff f(\vec{x}) = w, \tag{85}$$

and the next, simple proposition gives in many cases the easiest proofs of recursiveness for partial functions:

4A.6. PROPOSITION (**Graph Lemma**). *A partial function $f(\vec{x})$ is recursive if and only if its graph $G_f(\vec{x}, w)$ is a semirecursive relation.*

PROOF. If $f(\vec{x})$ is recursive with code $\widehat{f}$, then

$$G_f(\vec{x}, w) \iff (\exists y)[T_n(\widehat{f}, \vec{x}, y) \ \& \ U(y) = w],$$

so that $G_f(\vec{x}, w)$ is semirecursive; and if

$$f(\vec{x}) = w \iff (\exists u) R(\vec{x}, w, u)$$

with some recursive $R(\vec{x}, w, u)$, then

$$f(\vec{x}) = \Big(\mu u R(\vec{x}, (u)_0, (u)_1)\Big)_0,$$

so that $f(\vec{x})$ is recursive. ⊣

The last result in this section simplifies significantly many constructions.

4A.7. PROPOSITION (Σ^0_1**-Selection Lemma**). *For every semirecursive relation $R(\vec{x}, w)$, there is a recursive partial function $f(\vec{x})$ such that for all $\vec{x}$,*

$$\begin{aligned} (\exists w) R(\vec{x}, w) &\iff f(\vec{x})\downarrow \\ (\exists w) R(\vec{x}, w) &\implies R(\vec{x}, f(\vec{x})). \end{aligned}$$

PROOF. By the hypothesis, there exists a recursive relation $P(\vec{x}, w, y)$ such that

$$R(\vec{x}, w) \iff (\exists y) P(\vec{x}, w, y), \tag{$**$}$$

and the conclusion of the Lemma follows easily if we set

$$f(\vec{x}) = \Big(\mu u P(\vec{x}, (u)_0, (u)_1)\Big)_0. \qquad \dashv$$

It is suggestive and sometimes convenient to use the notation

$$f(\vec{x}) = (\nu w) R(\vec{x}, w) \tag{86}$$

for the partial function $f(\vec{x})$ where ν is read as *some*; but it is important to remember that $f(\vec{x})$ is not uniquely determined by the semirecursive relation $R(\vec{x})$, it needs a representation of $R(\vec{x})$ as in $(**)$ for its definition.

Problems for Section 4A

x4A.1. Prove that the partial function

$$f(e,u) = \langle \varphi_e((u)_0), \dots, \varphi_e((u)_{\mathrm{lh}(u) \dot{-} 1}) \rangle$$

is recursive.

x4A.2. Let $R(\vec{x}, w)$ be a semirecursive relation such that for every $\vec{x}$ there exist at least two numbers $w_1 \neq w_2$ such that $R(\vec{x}, w_1)$ and $R(\vec{x}, w_2)$. Prove that there exist two, total recursive functions $f(\vec{x})$, $g(\vec{x})$ such that for all $\vec{x}$,

$$R(\vec{x}, f(\vec{x})) \ \& \ R(\vec{x}, g(\vec{x})) \ \& \ f(\vec{x}) \neq g(\vec{x}).$$

x4A.3*. Let $R(\vec{x}, w)$ be a semirecursive relation such that for every $\vec{x}$, there exists at least one w such that $R(\vec{x}, w)$.

(1) Prove that there exists a total recursive function $f(n, \vec{x})$, such that

$$R(\vec{x}, w) \iff (\exists n)[w = f(n, \vec{x})]. \tag{87}$$

(2) Prove that if, in addition, for every $\vec{x}$, there exist infinitely many w such that $R(\vec{x}, w)$, then there exists a total, recursive $f(n, \vec{x})$ which satisfies (87) and is "1-1 in n", i.e., for all $\vec{x}, m, n$,

$$m \neq n \Longrightarrow f(m, \vec{x}) \neq f(n, \vec{x}).$$

4B. Recursively enumerable sets

4B.1. Definition. A set $A \subseteq \mathbb{N}$ is **recursively enumerable** (r.e.), if $A = \emptyset$ or some recursive total function $f : \mathbb{N} \to \mathbb{N}$ enumerates A,

$$A = f[\mathbb{N}] = \{f(0), f(1), \dots\}. \tag{88}$$

4B.2. Proposition. (1) *The following are equivalent for any $A \subseteq \mathbb{N}$:*

(a) *A is r.e.*

(b) *The relation $x \in A$ is semirecursive, so that*

$$A = \mathrm{Domain}(g) = \{x \mid g(x)\downarrow\}$$

for some recursive partial function g.

(c) *A is finite, or there is a recursive injection $f : \mathbb{N} \rightarrowtail \mathbb{N}$ which enumerates A, $A = f[\mathbb{N}]$.*

(2) *The sequence*

$$W_0, W_1, \dots$$

enumerates the r.e. sets, so that the relation $x \in W_e$ is semirecursive.

(3) *A set $A \subseteq \mathbb{N}$ is recursive if and only if it is finite or there exists a* (strictly) *increasing, recursive function which enumerates A,*

$$A = \{f(0) < f(1) < f(2) < \cdots\}.$$

Proof. For (3), first, we recall (as in x1A.12) that a function $f : \mathbb{N} \to \mathbb{N}$ is strictly increasing if

$$f(n) < f(n+1) \qquad (n \in \mathbb{N}),$$

which implies immediately (by induction) that

$$n \leq f(n);$$

it follows that if A is enumerated by an increasing, recursive f, then

$$x \in A \iff (\exists n \leq x)[x = f(n)],$$

and A is recursive. For the other direction, if A is recursive and infinite, then the function

$$f(0) = (\mu x)[x \in A]$$
$$f(n+1) = (\mu x)[x > f(n) \ \&\ x \in A]$$

is recursive, increasing and enumerates A.

(1) The implication (a) $\Longrightarrow$ (b) is trivial for $A = \emptyset$, and if $A = f[\mathbb{N}]$, then

$$x \in A \iff (\exists i)[x = f(i)].$$

The converse (b) $\Longrightarrow$ (a) is also trivial for $A = \emptyset$, and if $x_0 \in A$ and

$$x \in A \iff (\exists y)R(x, y),$$

then A is enumerated by the recursive total function

$$f(u) = \begin{cases} x_0, & \text{if } \neg R((u)_0, (u)_1), \\ (u)_0, & \text{if } R((u)_0, (u)_1). \end{cases}$$

Finally, the implication (c) $\Longrightarrow$ (a) is trivial, as is (a) $\Longrightarrow$ (c) for finite A.

It remains to show that if A is infinite and

$$A = \{f(0), f(1), \dots\}$$

for some recursive function f, then A is also enumerated by some one-to-one recursive function. The basic idea is to "delete the repetitions" from the enumeration by f, something that *obviously* leads to a total, one-to-one *recursive* enumeration of A. For the rigorous proof of this proposition, let

$$B = \{n \mid (\forall i < n)[f(i) \neq f(n)]\}$$

be the recursive set of the positions where new members of A are enumerated by f; then $B = g[\mathbb{N}]$ for some increasing $g(n)$ by part (3), and A is enumerated by the composition $h(n) = f(g(n))$, which is the composition of two injections and hence injective.

Figure 5. Deletion of repetitions.

Part (2) follows by the characterization (b) of (1). ⊣

4B.3. Corolary. *Every recursive set is r.e., but there exist r.e. sets which are not recursive, e.g.,*

$$H' = \{x \mid H((x)_0, (x)_1)\}. \tag{89}$$

Proof. If H' were recursive then the halting relation (73) would also be recursive, since

$$H(e,x) \iff \langle e,x\rangle \in H'.$$ ⊣

The class of r.e. sets has a very rich structure and it has been studied intensely. Here we will confine ourselves to (very few) basic results, which give an idea of its properties.

4B.4. Definition. A **reduction** of a set A to B, is any (total) recursive function f which satisfies the equivalence

$$x \in A \iff f(x) \in B. \tag{90}$$

For any two sets $A, B \subseteq \mathbb{N}$, we set:

$A \leq_m B \iff$ there exists a reduction of A to B,

$A \leq_1 B \iff$ there exists a one-to-one reduction of A to B,

$A \equiv B \iff$ there exists a reduction of A to B which is a permutation,

where $f : \mathbb{N} \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{N}$ is a *permutation* if it is a bijection, one-to-one and onto $\mathbb{N}$. Obviously,

$$A \equiv B \Longrightarrow A \leq_1 B \Longrightarrow A \leq_m B.$$

4B.5. Proposition. *For all sets A, B, C,*

$$A \leq_m A \text{ and } [A \leq_m B \ \&\ B \leq_m C] \Longrightarrow A \leq_m C,$$

and the same holds for the stronger reductions $\leq_1$ and $\equiv$; moreover, the relation of recursive isomorphism $\equiv$ is symmetric,

$$A \equiv B \iff B \equiv A.$$

Proof. For the transitivity of these relations, we notice that if, by the hypothesis

$$x \in A \iff g(x) \in B \text{ and } y \in B \iff h(y) \in C,$$

for given recursive functions, then

$$x \in A \iff h(g(x)) \in C,$$

so that the composition $f(x) = h(g(x))$ reduces A to C. ⊣

4B.6. Definition. A set B is **r.e. complete** if it is r.e. and each r.e. A is reducible to B by a recursive injection, $A \leq_1 B$.

For example, the set H' that we defined in (89) is complete, because each r.e. set is of the form W_e for some e, and

$$x \in W_e \iff \varphi_e(x)\downarrow \iff \langle e, x\rangle \in H'.$$

A little simpler is Post's "diagonal" set

$$K = \{x \mid (\exists y)T_1(x, x, y)\} = \{x \mid \varphi_x(x)\downarrow\}, \tag{91}$$

whose completeness is not quite immediate:

4B.7. Proposition. *The set K is r.e. complete.*

Proof. For any r.e. set

$$A = \{x \mid g(x)\downarrow\}$$

(with recursive $g(x)$), set

$$h(x, y) = g(x)$$

and choose some code $\widehat{h}$ of h, so that for any y,

$$\begin{aligned} x \in A &\iff h(x, y)\downarrow \\ &\iff \{\widehat{h}\}(x, y)\downarrow \\ &\iff \{S^1_1(\widehat{h}, x)\}(y)\downarrow . \end{aligned}$$

This equivalence holds for every y, so in particular it holds for $y = S^1_1(\widehat{h}, x)$ and gives

$$\begin{aligned} x \in A &\iff \{S^1_1(\widehat{h}, x)\}(S^1_1(\widehat{h}, x))\downarrow \\ &\iff S^1_1(\widehat{h}, x) \in K, \end{aligned}$$

which reduces A to K by the injection $f(x) = S^1_1(\widehat{h}, x)$. ⊣

The next, basic theorem shows in part that up to recursive isomorphism there exists only one r.e. complete set. It is, however, much stronger than this: it holds for all sets A, B, not only for recursively enumerable sets—and this is what makes the proof difficult.

4B.8. Theorem (**Myhill's Theorem**). *For any two sets A, B,*

$$A \leq_1 B \;\&\; B \leq_1 A \Longrightarrow A \equiv B.$$

Proof. The argument is a constructive version of the classical Schröder-Bernstein Theorem about sets, and it is based on two lemmas about non-empty sequences of pairs

$$W = (x_0, y_0), (x_1, y_1), \dots, (x_n, y_n). \tag{92}$$

A sequence W of this form is *injective* if

$$i \neq j \Longrightarrow [x_i \neq x_j \ \& \ y_i \neq y_j] \quad (i, j \leq n),$$

and *good* (as an approximation to an isomorphism) for A and B if it is injective and in addition

$$x_i \in A \iff y_i \in B \qquad (i \leq n).$$

For any sequence of pairs W as in (92), we set

$$X = \{x_0, x_1, \dots, x_n\}, \quad Y = \{y_0, y_1, \dots, y_n\}.$$

Lemma X. *If $A \leq_1 B$, then for every injective sequence* (92) *and each $x \notin X$, we can find some $y \notin Y$ such that the extension*

$$W' = (x_0, y_0), (x_1, y_1), \dots, (x_n, y_n), (x, y) \tag{93}$$

is injective, and if W is good, then W' is also good.

Prof of Lemma X. The hypothesis gives us a recursive one-to-one function $f : \mathbb{N} \rightarrowtail \mathbb{N}$ such that

$$x \in A \iff f(x) \in B.$$

We define by recursion

$$z_0 = f(x)$$
$$z_{i+1} = \begin{cases} z_i & \text{if } z_i \notin Y, \\ f(x_j) & \text{otherwise, if } z_i = y_j, \end{cases}$$

and we verify two basic properties of the sequence $z_0, z_1, \dots$.

(1) *If W is injective, then*

$z_i \in Y \Longrightarrow z_0, z_1, \dots, z_i$ are all distinct and $\{z_0, \dots, z_i\} \subseteq f[X \cup \{x\}]$.

Proof is by induction on i, and it is obvious at the basis since $z_0 = f(x)$. At the induction step, we assume that $z_{i+1} \in Y$. This implies that $z_i \in Y$; because if $z_i \notin Y$, then $z_{i+1} = z_i$ by the definition, which contradicts the assumption that $z_{i+1} \in Y$. So the induction hypothesis assures us that $z_0, z_1, \dots, z_i$ are all distinct and lie in $f[X \cup \{x\}]$. It suffices to prove that z_{i+1} is not in $\{z_0, \dots, z_i\}$. Notice that $z_{i+1} \neq z_0$, since $z_0 = f(x)$, $z_{i+1} = f(x_j)$ for some j, $x \notin X$ and f is an injection. So it suffices to derive a contradiction from the assumption that

$$z_{i+1} = z_{k+1} \text{ for some } k < i,$$

and the definition gives us that

$$z_{i+1} = f(x_j) \text{ where } y_j = z_i \text{ and } z_{k+1} = f(x_s) \text{ where } z_k = y_s.$$

Using this and the hypotheses that f and W are injective, we have

$$z_{i+1} = z_{k+1} \Longrightarrow (f(x_j) = f(x_s)) \Longrightarrow (x_j = x_s) \Longrightarrow (y_j = y_s) \Longrightarrow z_i = z_k;$$

and this contradicts the induction hypothesis.

Now (1) implies that for some $j < n+2$, $z_j \notin Y$ (since Y has $n+1$ members), and the first claim in the Lemma holds if we set $y = z_j$ for the least such j.

(2) *If W is good, then for each i, $x \in A \iff z_i \in B$.*

Proof. For $i = 0$, $x \in A \iff f(x) = z_0 \in B$, by the hypothesis on f. Inductively, if $z_i \notin Y$ with $i > 0$, then

$$x \in A \iff z_{i+1} = z_i \in B$$

by the induction hypothesis, and if $z_i \in Y$, then

$$\begin{aligned} x \in A &\iff z_i = y_j \in B && \text{(for some } j\text{, by the induction hypothesis)} \\ &\iff x_j \in A && \text{(because the given sequence is good)} \\ &\iff f(x_j) = z_{i+1} \in B. \end{aligned}$$

This completes the proof of the Lemma ⊣

The symmetric Lemma Y gives us for each injective sequence W and each $y \notin Y$ some $x \notin X$ such that the extension $W' = W, (x, y)$ is injective and also good, if W is good. The construction of the required recursive permutation proceeds by successive application of these two Lemmas starting with the good sequence

$$W_0 = \langle 0, f(0)\rangle, \quad X_0 = \{0\}, Y_0 = \{f(0)\}.$$

Odd step $2n+1$. Let $y = \min(\mathbb{N} \setminus Y_{2n})$ and extend W_{2n} by applying Lemma Y, so that $y \in Y_{2n+1}$.

Even step $2n+2$. Let $x = \min(\mathbb{N} \setminus X_{2n+1})$ and extend W_{2n+1} by applying Lemma X so that $x \in X_{2n+2}$.

In the end, the union $\bigcup_n W_n$ is the graph of a permutation $h : \mathbb{N} \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{N}$ which reduces A to B,

$$x \in A \iff h(x) \in B.$$

The recursiveness of h follows from the construction and completes the proof that $A \equiv B$. ⊣

4B.9. **Proofs of undecidability**. If $A \leq_m B$, and B is recursive, then A is also recursive; it follows that if $A \leq_m B$ and A is not recursive, then B is not recursive either. Together with the completeness of K, this simple proposition is the first, basic tool for the proof of non-recursiveness of sets

and relations: because if we show that $A \leq_m B$ with some A which is not recursive (e.g., K), then it follows that B is not recursive.

4B.10. Proposition (Example). *The set*

$$A = \{e \mid W_e \neq \emptyset\}$$

is r.e. but not recursive.

Proof. The set A is r.e. because the relation

$$e \in A \iff (\exists x)[x \in W_e]$$

is Σ^0_1. To show that $K \leq_1 A$, we set

$$g(e,x) = \mu y T_1(e,e,y)$$

so that the value $g(e,x)$ is independent of x,

$$g(e,x) = \begin{cases} \mu y T_1(e,e,y), & \text{if } (\exists y) T_1(e,e,y), \\ \uparrow, & \text{otherwise,} \end{cases}$$

and, for every x,

$$e \in K \iff g(e,x)\downarrow,$$

so that

$$e \in K \iff (\exists x) g(e,x)\downarrow;$$

it follows that if $\widehat{g}$ is a code of $g(x,y)$, then

$$\begin{aligned} e \in K &\iff (\exists x)[\{\widehat{g}\}(e,x)\downarrow] \\ &\iff (\exists x)[\{S^1_1(\widehat{g},e)\}(x)\downarrow] \\ &\iff W_{S^1_1(\widehat{g},e)} \neq \emptyset \\ &\iff S^1_1(\widehat{g},e) \in A, \end{aligned}$$

so $K \leq_1 A$ and A is not recursive. ⊣

We notice that with this construction,

$$\begin{aligned} e \in K &\iff W_{S^1_1(\widehat{g},e)} = \mathbb{N} \\ &\iff W_{S^1_1(\widehat{g},e)} \text{ has at least 2 members} \end{aligned}$$

so that the sets

$$B = \{e \mid W_e = \mathbb{N}\}, \quad C = \{e \mid W_e \text{ has at least 2 members}\}$$

are not recursive either.

4B.11. A **recursive separation** of two sets A and B is any recursive set C such that $A \subseteq C$ and $B \cap C = \emptyset$.

If $A \cap B \neq \emptyset$, then, obviously, there is no separation of A from B, and if A is recursive and $A \cap B = \emptyset$, then, obviously again, A separates itself from B.

4B.12. PROPOSITION (Kleene). *There exist r.e. sets K_0 and K_1 such that $K_0 \cap K_1 = \emptyset$ and there is no recursive separation of K_0 from K_1.*

PROOF. We set

$$K_0 = \{e \mid (\exists y)[T_1((e)_0, e, y) \ \&\ (\forall z \leq y)\neg T_1((e)_1, e, z)]\},$$
$$K_1 = \{e \mid (\exists z)[T_1((e)_1, e, z) \ \&\ (\forall y < z)\neg T_1((e)_0, e, y)]\},$$

and we show first that $K_0 \cap K_1 = \emptyset$ by contradiction: because if $e \in K_0 \cap K_1$ and

$$y^* = \mu y[T_1((e)_0, e, y) \ \&\ (\forall z \leq y)\neg T_1((e)_1, e, z)]$$
$$z^* = \mu z[T_1((e)_1, e, z) \ \&\ (\forall y < z)\neg T_1((e)_0, e, y)],$$

then, by the definitions,

$$\begin{aligned} y^* < z^* &\Longrightarrow T_1((e)_0, e, y^*) \ \&\ (\forall y < z^*)\neg T_1((e)_0, e, y) \\ &\Longrightarrow T_1((e)_0, e, y^*) \ \&\ \neg T_1((e)_0, e, y^*), \end{aligned}$$

and the opposite hypothesis $z^* \leq y^*$ also gives a contradiction by a similar argument.

To show that K_1 and K_2 are recursively inseparable, again by contradiction, suppose W_e, W_m r.e. sets such that

$$K_0 \subseteq W_e, \quad K_1 \subseteq W_m, \quad W_e \cap W_m = \emptyset, \quad W_e \cup W_m = \mathbb{N},$$

and let $t = \langle m, e\rangle$; we compute:

$$\begin{aligned} t \in W_e &\Longrightarrow (\exists z)T_1(e, \langle m, e\rangle, z) \ \&\ (\forall y)\neg T_1(m, \langle m, e\rangle, y) \\ &\qquad \text{because } W_e \cap W_m = \emptyset \\ &\Longrightarrow (\exists z)[T_1(e, \langle m, e\rangle, z) \ \&\ (\forall y < z)\neg T_1(m, \langle m, e\rangle, y)] \\ &\Longrightarrow \langle m, e\rangle \in K_1 \qquad \text{by the definition} \\ &\Longrightarrow \langle m, e\rangle \in W_m \qquad \text{because } K_1 \subseteq W_m, \end{aligned}$$

so $t \in W_e \cap W_m$ which is absurd. It follows that $t \in W_m$, since $W_e \cup W_m = \mathbb{N}$, but the symmetric computation derives again from this that $t \in W_m \cap W_e$, which is absurd. ⊣

4B.13. **Turing reducibility.** A set A is *Turing reducible* to a set B if its characteristic function χ_A is recursive in the characteristic function χ_B of B, in symbols

$$A \leq_T B \iff \chi_A \in \mathcal{R}(\mathbf{N}_0, \chi_B). \tag{94}$$

Quite obviously,

$$A \leq_m B \Longrightarrow A \leq_T B,$$

so Turing reducibility is weaker than the reducibilities we have been studying. It is also the most natural: $A \leq_T B$ means that that some algorithm

with access to all the values of χ_B can decide whether $x \in A$ or not. The corresponding *equivalence relation*

$$A \equiv_T B \iff A \leq_T B \ \& \ B \leq_T A.$$

is the most natural notion of "recursive equivalence" for sets. This is trivially a reflexive and transitive relation, Problem x4B.12. Its equivalence classes are the *Turing degrees* (of unsolvability)

$$\deg_T(A) = \{B \mid B \equiv_T A\},$$

and the preordering $\leq_T$ induces on them the partial ordering

$$\begin{aligned} \mathbf{a} \leq_T \mathbf{b} &\iff (\exists A \in \mathbf{a}, B \in \mathbf{b})[A \leq_T B] \\ &\iff (\forall A \in \mathbf{a}, B \in \mathbf{b})[A \leq_T B]. \end{aligned}$$

We also set

$$\mathbf{0} = \deg(\emptyset) \ (= \deg(A) \text{ for every recursive } A), \quad \mathbf{0}' = \deg(K),$$

where K is the canonical, complete recursively enumerable set (91).

The partial ordering $\leq_T$ on the set $\mathcal{D}$ of *Turing degrees* has been and still is one of the most extensively studied objects of recursion theory since it was introduced by Emil Post in 1944. We will not study it in these notes, but it is worth stating three of the most basic results about it.

4B.14. Theorem. (1) *For every set $A \subseteq \mathbb{N}$ there exists some B with higher degree of unsolvability,* $\deg(A) <_T \deg(B)$.

(2) (Kleene-Post). *There exist sets $A, B \subseteq \mathbb{N}$ which are incomparable with respect to $\leq_T$, i.e., $A \not\leq_T B$ and $B \not\leq_T A$.*

(3) (Friedberg-Muchnik). *There exist r.e. sets $A, B \subseteq \mathbb{N}$ which are incomparable with respect to $\leq_T$, i.e., $A \not\leq_T B$ and $B \not\leq_T A$, therefore also*

$$\mathbf{0} <_T \deg(A) <_T \mathbf{0}'$$

and the same for B.

The improvement (3) of (2) is stated separately because it was an open problem ("Post's Problem") from 1944 until its solution in 1956. The *priority method* by which it was finally solved is still used extensively and is one of the basic methods of proof in computability theory.

Problems for Section 4B

x4B.1. Prove that there is a recursive, partial function $f(e)$, such that

$$W_e \neq \emptyset \Longrightarrow [f(e)\downarrow \ \& \ f(e) \in W_e].$$

x4B.2. For each of the following propositions decide whether it is true for arbitrary, recursively enumerable sets A and B. Prove your positive answers and give counterexamples for the negative ones.

(1) $A \cap B$ is r.e.
(2) $A \cup B$ is r.e.
(3) $A \setminus B = \{x \in A \mid x \notin B\}$ is r.e.

x4B.3. Prove that every infinite r.e. set has an infinite, recursive subset.

x4B.4. Are the following claims true or false? Give proofs or counterexamples.

(1) There is a (total) recursive function $u_1(e, m)$ such that for all e, m,
$$W_{u(e,m)} = W_e \cup W_m.$$
(2) There is a (total) recursive function $u_2(e, m)$ such that for all e, m,
$$W_{u(e,m)} = W_e \cap W_m.$$
(3) There is a (total) recursive function $u_3(e, m)$ such that for all e, m,
$$W_{u(e,m)} = W_e \setminus W_m.$$

x4B.5. Does there exist a total, recursive function $f(e, m)$ such that for all e, m,
$$W_{f(e,m)} = \{x + y \mid x \in W_e \text{ and } y \in W_m\}?$$
You must prove your answer.

x4B.6. Let $f : \mathbb{N} \to \mathbb{N}$ be a (total) recursive function, $A \subseteq \mathbb{N}$ and let
$$f[A] = \{f(x) \mid x \in A\}$$
$$f^{-1}[A] = \{x \mid f(x) \in A\}$$
be the *image* and the *inverse image* of A by f. For each one of the following claims decide whether it is true or not, prove your positive answers and give counterexamples for the negative ones.

(1) If A is r.e., is $f[A]$ also r.e.?
(2) If A is r.e., is $f^{-1}[A]$ also r.e.?
(3) If A is recursive, is $f[A]$ also recursive?
(4) If A is recursive, is $f^{-1}[A]$ also recursive?

x4B.7*. The *closure* $\overline{A}$ of a set $A \subseteq \mathbb{N}$ under a partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ is the smallest set B such that $B \supseteq A$ and B is closed for f, i.e.,
$$[x \in B \ \&\ f(x)\!\downarrow] \Longrightarrow f(x) \in B.$$

(1) Prove that if A is r.e. and $f(x)$ is recursive, then the closure $\overline{A}$ of A under f is also r.e.

(2) Prove that there exists a primitive recursive function $u(e, m)$, such that for all e and m, the set $W_{u(e,m)}$ is the closure $\overline{W}_e$ of W_e under the recursive partial function φ_m with code m.

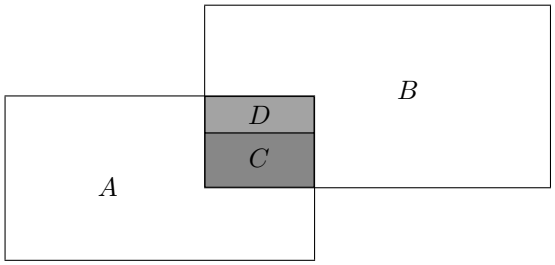
x4B.8. Prove that the set

$$K_0 = \{e \mid (\exists y)[T_1((e)_0, e, y) \ \& \ (\forall z \leq y)\neg T_1((e)_1, e, z)]\}$$

is r.e.-complete.

x4B.9. (The **reduction property** for the class of r.e. sets). Prove that if A and B are r.e., then there exist r.e. sets A_1, B_1, such that

$$A_1 \subseteq A, \quad B_1 \subseteq B, \quad A_1 \cup B_1 = A \cup B, \quad A_1 \cap B_1 = \emptyset.$$



$$A_1 = (A \setminus B) \cup C, \quad B_1 = (B \setminus A) \cup D$$

x4B.10 (The **separation property** for the class of r.e. complements). Prove that if A and B are complements of some r.e. sets A^c and B^c and $A \cap B = \emptyset$, then there exists a recursive C which separates A from B, i.e.,

$$A \subseteq C, \quad C \cap B = \emptyset.$$

Hint. Apply the preceding Problem to the complements A^c and B^c.

x4B.11. One of the two following propositions is true while the other is not. Give a proof of the true one and a counterexample of the one which is not true.

(1) If $A \subseteq B$ and A, B^c are r.e., then there exists a recursive set C such that $A \subseteq C \subseteq B$.

(2) If $A \subseteq B$ and A^c, B are r.e., then there exists a recursive set C such that $A \subseteq C \subseteq B$.

x4B.12. Prove that $A \leq_T B \ \& \ B \leq_T C \Longrightarrow A \leq_T C$.

4C. Productive, creative and simple sets

Up until now, the only r.e., non-recursive sets that we have met are r.e. complete, and the question arises whether every r.e. set is either recursive or r.e. complete. The question was asked by Emil Post in his 1944 paper which introduced Turing reducibility, and it was his main motivation for posing *Post's Problem*, whether there exist Turing-incomparable r.e. sets. He could not prove Part (3) of Theorem 4B.14 which solves this problem, but he was able to show that there are r.e. sets which are neither recursive nor r.e. complete using the stronger reducibility $A \leq_1 B$. We present here his constructions, which have applications beyond solving the basic problem for which they were introduced.

4C.1. Definition. A function $p : \mathbb{N} \rightarrowtail \mathbb{N}$ is a **productive function** for a set B if it is recursive, one-to-one and

$$W_e \subseteq B \Longrightarrow p(e) \in B \setminus W_e;$$

and B is **productive** if it has a productive function.

A set A is **creative** if it is r.e. and its complement

$$A^c = \{x \in \mathbb{N} \mid x \notin A\}$$

is productive.

4C.2. Proposition. *The set K is creative, with productive function for K^c the identity $p(e) = e$.*

Proof. We must show that

$$W_e \subseteq K^c \Longrightarrow e \in K^c \setminus W_e,$$

i.e.,

$$(\forall t)[t \in W_e \Longrightarrow t \notin K] \Longrightarrow [e \notin K \ \& \ e \notin W_e].$$

The hypothesis of the implication is

$$(\forall t)[\{e\}(t)\downarrow \ \Longrightarrow \{t\}(t)\uparrow]$$

and the conclusion is simply

$$\{e\}(e)\uparrow,$$

because

$$e \notin K \iff e \notin W_e \iff \{e\}(e)\uparrow;$$

and the hypothesis implies the conclusion, because if $\{e\}(e)\downarrow$, then setting $t = e$ in the hypothesis we have $\{e\}(e)\uparrow$, which is absurd. ⊣

4C.3. Corolary. *Every r.e. complete set is creative.*

We leave the proof for Problem, x4C.1.

The converse of this corollary also holds and gives a "structural" characterization of r.e. completeness, but its proof is not so simple and we will put it off until the next section. In the remainder of this section, we will construct non-recursive r.e. sets which are not creative, and hence not complete.

4C.4. Proposition. *Every productive set B has an infinite, r.e. subset.*

Proof. The idea is to define a function $f : \mathbb{N} \to \mathbb{N}$ by the recursion

$$\begin{aligned} f(0) &= e_0, \text{ where } W_{e_0} = \emptyset \\ f(x+1) &= \text{some code of } W_{f(x)} \cup \{p(f(x))\} \end{aligned}$$

where $p(e)$ is the given productive function for B. If we achieve this, then by an easy induction we can check that for every x,

$$W_{f(x)} \subsetneq W_{f(x+1)} \subseteq B,$$

so that the set

$$A = W_{f(0)} \cup W_{f(1)} \cup \ldots = \{y \mid (\exists x)[y \in W_{f(x)}\}$$

is an r.e., infinite subset of B. For the computation of the required $h(w, x)$ such that

$$f(x+1) = h(f(x), x),$$

we set first

$$R(e, y, x) \iff x \in W_e \vee x = y.$$

This is a semirecursive relation, and so for some $\widehat{g}$,

$$\begin{aligned} x \in W_e \cup \{y\} &\iff \{\widehat{g}\}(e, y, x)\downarrow \\ &\iff \{S^2_1(\widehat{g}, e, y)\}(x)\downarrow, \end{aligned}$$

which means that if we set

$$u(e, y) = S^2_1(\widehat{g}, e, y),$$

then

$$W_{u(e,y)} = W_e \cup \{y\}.$$

Finally we define

$$h(w, x) = u(w, p(w)),$$

and in the definition of f, we put

$$f(x+1) = h(f(x), x) = u(f(x), p(f(x))).$$

It follows that

$$W_{f(x+1)} = W_{f(x)} \cup \{p(f(x))\}$$

as the proof required. ⊣

4C.5. Definition. A set A is **simple**, if it is r.e. and its complement A^c is infinite and does not have an infinite, r.e. subset, i.e.,

$$W_e \cap A = \emptyset \Longrightarrow W_e \text{ is finite.}$$

4C.6. Theorem (Emil Post). *There exists a simple set.*

Proof. The relation

$$R(x,y) \iff y \in W_x \ \&\ y > 2x$$

is semirecursive, so by the Σ^0_1-Selection Lemma 4A.7 there exists a recursive, partial function $f(x)$ such that

$$\begin{aligned}(\exists y)[y \in W_x \ \&\ y > 2x] &\iff f(x)\downarrow \\ &\iff f(x)\downarrow \ \&\ f(x) \in W_x \ \&\ f(x) > 2x.\end{aligned}$$

The required set is the image of f,

$$\begin{aligned}A = \{f(x) \mid f(x)\downarrow\} &= \{y \mid (\exists x)[f(x) = y]\} \\ &= \{y \mid (\exists x)[f(x) = y \ \&\ 2x < y]\},\end{aligned} \tag{95}$$

where the last equation follows from the definition of the relation $R(x,y)$.

(1) The set A is semirecursive by its definition, because the graph of $f(x)$ is Σ^0_1.

(2) The complement A^c is infinite, because

$$\begin{aligned}y \in A \ \&\ y \leq 2z \Longrightarrow (\exists x)[y = f(x) \ \&\ 2x < y \leq 2z] \\ \Longrightarrow (\exists x)[y = f(x) \ \&\ x < z],\end{aligned}$$

which implies that *at most z from the $2z+1$ numbers $\leq 2z$ belong to A*; it follows that some $y \geq z$ belongs to the complement A^c, and since this holds for every z, A^c is infinite.

(3) For each infinite W_e, $W_e \cap A \neq \emptyset$, because

$$\begin{aligned}W_e \text{ infinite} &\Longrightarrow (\exists y)[y \in W_e \ \&\ y > 2e] \\ &\Longrightarrow f(e)\downarrow \ \&\ f(e) \in W_e \\ &\Longrightarrow f(e) \in W_e \cap A.\end{aligned}$$

⊣

4C.7. Corolary. *Simple sets are neither recursive nor r.e. complete; so there exists an r.e., non-recursive set which is not r.e. complete.*

Proof. A simple set A cannot be recursive, because then its infinite complement would be r.e. without intersecting A; and it cannot be r.e. complete, because it is not creative by Proposition 4C.4. ⊣

Problems for Section 4C

x4C.1. Prove that if A is creative, B is r.e. and $A \leq_1 B$, then B is also creative.

x4C.2. Prove that if A is simple and B is r.e., infinite, then the intersection $A \cap B$ is infinite.

x4C.3*. Prove that if A and B are simple sets, then their intersection $A \cap B$ is also simple.

x4C.4. For the next two claims, decide whether they are true or not, and justify your answer by a proof or a counterexample:

(1) For every infinite r.e. set A, there is a total, recursive function f, such that for every x,

$$f(x) > x \text{ and } f(x) \in A.$$

(2) For every r.e. set B with infinite complement, there is a total recursive function g, such that for every x,

$$g(x) > x \text{ and } g(x) \notin B.$$

x4C.5*. (1) Prove that if A is simple, $f(x)$ is total, recursive and one-to-one and the inverse image $f^{-1}[A]$ has infinite complement, then the set $f^{-1}[A]$ is simple.

(2) Prove that if we omit any one of the hypotheses *total*, *one-to-one*, *infinite complement of* $f^{-1}[A]$, then the conclusion of Part (1) does not necessarily hold.

4D. The 2nd Recursion Theorem

In this section we will prove a misleadingly simple theorem of Kleene which has surprisingly strong and unexpected consequences. We will use it here for just one, important application, another theorem of Myhill which identifies the r.e., complete and the creative sets, but we will also find it very useful later in Chapter 6.

4D.1. The 2nd Recursion Theorem. *For every recursive, partial function $f(z, \vec{x})$, there exists a number z^* such that for all $\vec{x}$,*

$$\varphi_{z^*}(\vec{x}) = \{z^*\}(\vec{x}) = f(z^*, \vec{x}). \tag{96}$$

In fact, there is a primitive recursive function $h(e)$ (which depends only on the length n of the list $\vec{x} = x_1, \ldots, x_n$) *such that if $f = \varphi_e$, then* (96) *holds with $z^* = h(e)$, i.e., for all $e, \vec{x}$),*

$$\varphi_{h(e)}(\vec{x}) = \varphi_e(h(e), \vec{x}). \tag{97}$$

The theorem gives immediately some simple propositions which show that the coding of recursive partial functions has many unexpected (and somehow peculiar) properties.

4D.2. Proposition (Examples). *There exist natural numbers z_1 – z_4 such that*

$$\varphi_{z_1}(x) = z_1, \quad \varphi_{z_2}(x) = z_2 + x, \quad W_{z_3} = \{z_3\}, \quad W_{z_4} = \{0, \dots, z_4\}.$$

Proof. For z_1, we apply the 2nd Recursion Theorem to the function

$$f(z, x) = z$$

and we set $z_1 = z^*$; it follows that

$$\varphi_{z_1}(x) = f(z_1, x) = z_1.$$

The rest of the proofs are similar and equally simple. ⊣

Proof of the 2nd Recursion Theorem. Let

$$g(z, \vec{x}) = f(S^1_n(z, z), \vec{x}).$$

This is a recursive partial function, and so the Normal Form Theorem supplies us with a code c of it, so that

$$\{S^1_n(c, z)\}(\vec{x}) = \boxed{\{c\}(z, \vec{x}) = g(z, \vec{x})} = f(S^1_n(z, z), \vec{x}).$$

The conclusion of the 2nd Recursion Theorem follows from this equation if we set

$$z^* = S^1_n(c, c).$$

For the stronger version (97), choose d so that

$$\varphi_d(e, z, \vec{x}) = \varphi_e(S^1_n(z, z), \vec{x}).$$

By the S^m_n-Theorem,

$$c = S^1_{n+1}(d, e)$$

is a code of $\varphi_e(S^1_n(z, z), \vec{x})$, and the required function is:

$$h(e) = S^1_n(c, c) = S^1_n(S^1_{n+1}(d, e), S^1_{n+1}(d, e)).$$ ⊣

As a much more significant example of the strength of the 2nd Recursion Theorem, we show the converse of Corollary 4C.3, i.e., that every creative set is r.e. complete (and something more).

4D.3. Theorem (Myhill). *The following are equivalent for any r.e. set A:*

(1) *There exists a recursive partial function $p(e)$ such that*

$$W_e \cap A = \emptyset \Longrightarrow \Big(p(e)\downarrow \ \ \& \ p(e) \in A^c \setminus W_e\Big).$$

(2) *There exists a total recursive function $q(e)$ such that*

$$W_e \cap A = \emptyset \Longrightarrow q(e) \in A^c \setminus W_e. \tag{98}$$

(3) *A is creative, i.e.,* (98) *holds with a one-to-one recursive $q(e)$.*

(4) *A is r.e. complete.*

In particular, an r.e. set A is complete if and only if it is creative.

Proof. (1) $\Rightarrow$ (2). For the given, recursive partial function $p(e)$, there exists by the 2nd Recursion Theorem a number z such that

$$\{S_1^1(z,e)\}(t) = \varphi_z(e,t) = \begin{cases} \varphi_e(t), & \text{if } p(S_1^1(z,e))\downarrow, \\ \uparrow, & \text{otherwise.} \end{cases}$$

We set $q(e) = p(S_1^1(z,e))$ for this z and we notice that $q(e)$ is a total function, because

$$\begin{aligned} q(e) = p(S_1^1(z,e))\uparrow &\implies W_{S_1^1(z,e)} = \emptyset \quad \text{(by the definition)} \\ &\implies p(S_1^1(z,e))\downarrow . \end{aligned}$$

Moreover, since $q(e)\downarrow$, $W_{S_1^1(z,e)} = W_e$ by its definition, so

$$W_e \cap A = \emptyset \Longrightarrow q(e) = p(S_1^1(z,e)) \in A^c \setminus W_{S_1^1(z,e)} = A^c \setminus W_e$$

which is what we wanted to prove.

(2) $\Rightarrow$ (3) This implication does not need the 2nd Recursion Theorem and could have been given in Section 4C.

For the given function $q(e)$ which satisfies (98), we first notice that there exists a recursive partial function $h(e)$ such that

$$W_{h(e)} = W_e \cup \{q(e)\};$$

and then we set, recursively,

$$\begin{aligned} g(0,e) &= e \\ g(i+1,e) &= h(g(i,e)), \end{aligned}$$

so that (easily, by induction on i)

$$W_{g(i+1,e)} = W_e \cup \{q(g(0,e)), q(g(1,e)), \dots, q(g(i,e))\}.$$

It follows that for $i > 0$,

$$\begin{aligned} &W_e \cap A = \emptyset \\ &\Longrightarrow q(g(i,e)) \in A^c \setminus (W_e \cup \{q(g(0,e)), q(g(1,e)), \dots, q(g(i-1,e))\}), \end{aligned} \tag{99}$$

and, more specifically,

$$W_e \cap A = \emptyset \Longrightarrow (\forall j < i)[q(g(i,e)) \neq q(g(j,e))]. \tag{100}$$

Finally, we set

$$f(0) = q(0),$$

and for the (recursive) definition of $f(e+1)$, we first compute successively the values $q(g(0,e+1)), \dots, q(g(e+1,e+1))$ and we consider two cases.

Case 1. If these values are all distinct, then one of them is different from $f(0), \dots, f(e)$, and we set

$$j = (\mu i \leq (e+1))(\forall y \leq e)[q(g(i, e+1)) \neq f(y)]$$
$$f(e+1) = q(g(j, e+1)).$$

Case 2. There exist $i, j \leq e+1$, $i \neq j$, such that

$$q(g(i, e+1)) = q(g(j, e+1)).$$

In this case we set

$$f(e+1) = \max\{f(0), \dots, f(e)\} + 1.$$

It is obvious from the definition that $f(e)$ is recursive and one-to-one, and that it is a productive function for A^c follows immediately from (100) and (99).

(3) $\Rightarrow$ (4). If $q(e)$ is a productive function for A^c and B is any r.e. set, then by the 2nd Recursion Theorem there exists a number z such that

$$\varphi_z(x,t) = \begin{cases} 1, & \text{if } x \in B \ \&\ t = q(S^1_1(z,x)), \\ \uparrow, & \text{otherwise;} \end{cases}$$

the function $f(x) = q(S^1_1(z,x))$ is one-to-one, as a composition of injections, and reduces B to A, as follows.

If $x \in B$, then $W_{S^1_1(z,x)} = \{q(S^1_1(z,x)\} = \{f(x)\}$, and

$$\begin{aligned} f(x) \notin A &\Longrightarrow W_{S^1_1(z,x)} \cap A = \emptyset \\ &\Longrightarrow q(S^1_1(z,x)) \in A^c \setminus W_{S^1_1(z,x)} \\ &\Longrightarrow f(x) \in A^c \setminus \{f(x)\}, \end{aligned}$$

which is absurd; so $f(x) \in A$. On the other hand, if $x \notin B$, then $W_{S^1_1(z,x)} = \emptyset \subseteq A^c$, so $f(x) = q(S^1_1(z,x)) \in A^c$.

Finally, (4) $\Rightarrow$ (1) by Corollary 4C.3. $\dashv$

Problems for Section 4D

x4D.1. Prove that for some z, $W_z = \{z, z+1, \dots\} = \{x \mid x \geq z\}$.

x4D.2. Prove that for some z, $\varphi_z(t) = t \cdot z$.

x4D.3*. Are they true or not—and you must prove your answers:

(1) There is a recursive, partial function $f(e)$, such that for every e,

(*) if W_e is infinite, then $f(e)\downarrow$ & $f(e) \in W_e$ & $f(e) > e$.

(2) There exists a *total* recursive function $f(e)$ which satisfies (*).

x4D.4. Prove that for every recursive total function $f(z)$, either there exists some z such that $f(z)$ is odd, and for all x,

$$\varphi_z(x) = f(z + x),$$

or there exists some w such that $f(w)$ is even, and for all x,

$$\varphi_w(x) = f(2w + x + 1).$$

x4D.5. Determine whether it is true or false and prove your answer: for every recursive total function $f(z)$, there exists some z such that

$$W_{f(z)} = W_z.$$

x4D.6. Determine whether it is true or false and prove your answer: for every recursive total function $f(z)$, there exists a z such that

$$\varphi_{f(z)}(t) = \varphi_z(t) \quad (t \in \mathbb{N}).$$

x4D.7. (1) Prove that for every recursive total function $f(x)$, there exists some number z such that

$$W_z = \{f(z)\}.$$

(2) Prove that there exists some number z such that

$$\varphi_z(z)\downarrow \quad \text{and} \quad W_z = \{\varphi_z(z)\}.$$

x4D.8*. Let $g(e)$ be a recursive, partial function such that for all e,

$$\text{if } W_e = \mathbb{N}, \text{ then } g(e)\downarrow;$$

prove that there exist numbers m and k, such that

$$W_m = \{0, 1, \dots, k\} \text{ and } g(m)\downarrow .$$

Hint: Apply the 2nd Recursion Theorem to the partial function

$$f(m, x) = \begin{cases} 1, & \text{if } (\forall y \leq x)\neg T_1(\widehat{g}, m, y), \\ \uparrow, & \text{otherwise,} \end{cases}$$

where $g(e) = \{\widehat{g}\}(e)$.

x4D.9*. Let $g(e)$ be a recursive partial function such that for all e,

$$\text{if } W_e = \emptyset, \text{ then } g(e)\downarrow;$$

Prove that there is some m such that $W_m = \{m\}$ and $g(m)\downarrow$.

CHAPTER 5

RECURSION AND DEFINABILITY

In this chapter we will study those relations on the natural numbers which can be "constructed" starting with the recursive relations and applying repeatedly the non-effective operations of first-order logic, the quantifiers $\exists$ and $\forall$. Basic corollaries of this study are the classical theorems of Tarski, Gödel and Church in Section 5D, but there are many others, in logic, set theory and even mathematical analysis.

5A. The arithmetical hierarchy

A semirecursive relation $R(\vec{x})$ satisfies an equivalence

$$R(\vec{x}) \iff (\exists y)Q(\vec{x}, y)$$

with some recursive $Q(\vec{x}, y)$. To verify whether $R(\vec{x})$ holds for a particular $\vec{x}$ we need, in principle, to check all the decidable propositions

$$Q(\vec{x}, 0), Q(\vec{x}, 1) \dots ,$$

so we can say that the "distance in complexity" of $R(\vec{x})$ from the recursive $Q(\vec{x}, y)$ is just *one existential quantifier*. The next definition makes precise this intuitive notion of "distance from decidability" for relations on $\mathbb{N}$. It is the main tool for the classification of complex, undecidable relations.

5A.1. **The arithmetical classes.** The classes of relations Σ^0_k, Π^0_k, Δ^0_k are defined by the following recursion on $k \geq 1$:

$$\begin{aligned}
\Sigma^0_1 &: \text{the semirecursive relations} \\
\Pi^0_k = \neg\Sigma^0_k &: \text{the negations (complements) of relations in } \Sigma^0_k \\
\Sigma^0_{k+1} = \exists\Pi^0_k &: \text{the relations which satisfy an equivalence of the form} \\
&\qquad P(\vec{x}) \iff (\exists y)Q(\vec{x}, y) \text{ with some } Q(\vec{x}, y) \text{ is } \Pi^0_k \\
\Delta^0_k = \Sigma^0_k \cap \Pi^0_k &: \text{the relations which are both } \Sigma^0_k \text{ and } \Pi^0_k.
\end{aligned}$$

A set $A \subseteq \mathbb{N}$ is in one of these classes Γ if its membership relation $x \in A$ belongs to Γ.

Suppose, for example that $Q(x,y)$ is a recursive relation and put

$$R(x) \iff Q(x,y) \text{ is true for infinitely many } y's;$$

It follows that $R(x)$ is Π^0_2, since

$$R(x) \iff (\forall t)(\exists y)[R(x,y) \ \& \ y > t].$$

5A.2. **Canonical forms.** The *arithmetical classes* Σ^0_k, Π^0_k are easily characterized by the following "canonical forms", i.e., $P(\vec{x})$ belongs to one of these classes Γ if and only if it is equivalent with a relation in the canonical form for Γ with some recursive relation Q:

$$\begin{aligned}
\Sigma^0_1 &: && (\exists y)Q(\vec{x},y)\\
\Pi^0_1 &: && (\forall y)Q(\vec{x},y)\\
\Sigma^0_2 &: && (\exists y_1)(\forall y_2)Q(\vec{x},y_1,y_2)\\
\Pi^0_2 &: && (\forall y_1)(\exists y_2)Q(\vec{x},y_1,y_2)\\
\Sigma^0_3 &: && (\exists y_1)(\forall y_2)(\exists y_3)Q(\vec{x},y_1,y_2,y_3)\\
&\vdots
\end{aligned}$$

For example, if the relation $P(\vec{x})$ is Π^0_2, then, by the definitions,

$$\begin{aligned}
P(\vec{x}) &\iff \neg P_1(\vec{x}) && \text{with } P_1 \in \Sigma^0_2,\\
&\iff \neg(\exists y_1)P_2(\vec{x},y_1) && \text{with } P_2 \in \Pi^0_1,\\
&\iff \neg(\exists y_1)\neg P_3(\vec{x},y_1) && \text{with } P_3 \in \Sigma^0_1,\\
&\iff \neg(\exists y_1)\neg(\exists y_2)Q(\vec{x},y_1,y_2) && \text{with } Q \text{ recursive},\\
&\iff (\forall y_1)(\exists y_2)Q(\vec{x},y_1,y_2)
\end{aligned}$$

A full proof of this for every Σ^0_k, Π^0_k can be given easily by induction on k.

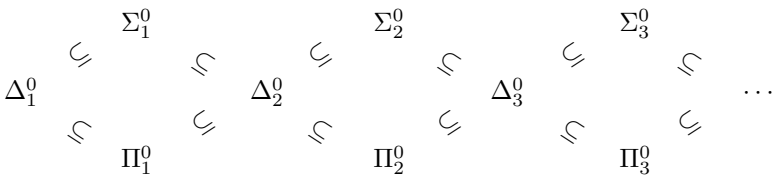
5A.3. Theorem. (1) *For every $k \geq 1$, the classes Σ^0_k, Π^0_k and Δ^0_k are closed under recursive substitutions and under the operators $\&$, $\vee$, $\exists_{\leq}$ and $\forall_{\leq}$. Moreover, for every $k \geq 1$:*

- *The class Δ^0_k is closed under negation $\neg$.*
- *The class Σ^0_k is closed under the existential quantifier $\exists$.*
- *The class Π^0_k is closed under the universal quantifier $\forall$.*

(2) *For every $k \geq 1$,*

$$\Sigma^0_k \subseteq \Delta^0_{k+1}, \tag{101}$$

and so the arithmetical classes satisfy the following diagram of inclusions:



Proof. First we show the closure of all arithmetical classes under recursive substitutions, by induction on k. This fact is known for $k = 1$ by Proposition 4A.5, and inductively (for the case Σ^0_{k+1}) we compute:

$$\begin{aligned} P(\vec{x}) &\iff R(f_1(\vec{x}), \dots, f_n(\vec{x})) \\ &\iff (\exists y) Q(f_1(\vec{x}), \dots, f_n(\vec{x}), y), \text{ with } Q \in \Pi^0_k, \text{ by the definition} \\ &\iff (\exists y) Q'(\vec{x}, y) \text{ with } Q' \in \Pi^0_k \text{ by the induction hypothesis.} \end{aligned}$$

The rest of Part (1) is easily checked, by induction on k and applications of the transformations in the proof of Proposition 4A.5.

Part (2) is also proved by induction on k, where, at the basis, if

$$P(\vec{x}) \iff (\exists y) Q(\vec{x}, y)$$

with a recursive Q, then P is certainly Σ^0_2, since every recursive relation is Π^0_1. It is also Π^0_2, since, obviously,

$$P(\vec{x}) \iff (\forall z)(\exists y) Q(\vec{x}, y)$$

and the relation

$$Q_1(\vec{x}, z, y) \iff Q(\vec{x}, y)$$

is recursive. The proof at the induction step is exactly the same, and the inclusions in the diagram follow easily from (101). ⊣

More interesting is the following theorem which justifies the name "hierarchy" for the classes Σ^0_k, Π^0_k:

5A.4. **The Arithmetical Hierarchy Theorem.** *For all $n, k \geq 1$:*

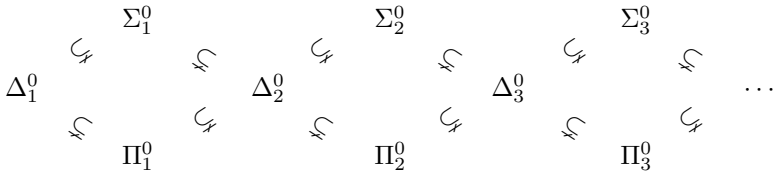
(1) (Enumeration for Σ^0_k) *There is an $(n+1)$-place relation $S_{k,n}(e, \vec{x})$ in the class Σ^0_k which enumerates the n-place, Σ^0_k relations: i.e., an n-ary relation $P(\vec{x})$ is Σ^0_k if and only if for some e,*

$$P(\vec{x}) \iff S_{k,n}(e, \vec{x}).$$

(2) (Enumeration for Π^0_k) *There is an $(n+1)$-place relation $P_{k,n}(e, \vec{x})$ in the class Π^0_k which enumerates the n-place, Π^0_k relations: i.e., an n-ary relation $P(\vec{x})$ is Π^0_k if and only if for some e,*

$$P(\vec{x}) \iff P_{k,n}(e, \vec{x}).$$

(3) (Hierarchy) *The inclusions in the diagram of Proposition* 5A.3 *are all strict, i.e.,*



Proof. For (1) and (2) we set recursively

$$S_{1,n}(e,\vec{x}) \iff (\exists y)T_n(e,\vec{x},y), \quad P_{k,n}(e,\vec{x}) \iff \neg S_{k,n}(e,\vec{x})$$
$$S_{k+1,n}(e,\vec{x}) \iff (\exists y)P_{k,n+1}(e,\vec{x},y),$$

and the proofs are easy, by induction on k. For (3), we observe that the *diagonal relation*

$$D_k(x) \iff S_{k,1}(x,x)$$

is Σ^0_k but cannot be Π^0_k; because if it were, then for some e we would have,

$$\neg S_{k,1}(x,x) \iff S_{k,1}(e,x)$$

which is absurd for $x = e$. It follows that for each k, there exist relations which are Σ^0_k but not Π^0_k, and from this follows easily the strictness of all inclusions in the diagram. ⊣

5A.5. A (complete) **classification** of a relation $P(\vec{x})$ in the arithmetical hierarchy is the determination of the "least" arithmetical class to which $P(\vec{x})$ belongs, i.e., the proof of a proposition of the form

$$P \in \Sigma^0_k \setminus \Pi^0_k \quad \text{or} \quad P \in \Pi^0_k \setminus \Sigma^0_k \quad \text{or} \quad P \in \Delta^0_{k+1} \setminus (\Sigma^0_k \cup \Pi^0_k)$$

for some k. For example, in 4B.10 we showed that

$$\{e \mid W_e \neq \emptyset\} \in \Sigma^0_1 \setminus \Pi^0_1.$$

The complete classification of a relation is in some cases very difficult, so we often settle for the computation of some "upper bound", namely some k such that $P \in \Sigma^0_k$ or $P \in \Pi^0_k$. The basic method for the computation of some "lower" bound, when this is feasible, is the proof that the given relation is *complete* in some class Σ^0_k or Π^0_k as in the next result.

5A.6. Proposition. (1) *The set* $F = \{e \mid \varphi_e \text{ is total}\}$ *is in* $\Pi^0_2 \setminus \Sigma^0_2$.
(2) *The set* $\mathrm{Fin} = \{e \mid W_e \text{ is finite}\}$ *is in* $\Sigma^0_2 \setminus \Pi^0_2$.

Proof. (1) The upper bound is obvious, since

$$e \in F \iff (\forall x)(\exists y)T_1(e,x,y).$$

To show by contradiction that F is not Σ^0_2, we consider any relation $P(x)$ in Π^0_2. By the definition, there is a recursive $Q(x,u,v)$ such that

$$P(x) \iff (\forall u)(\exists v)Q(x,u,v.)$$

Set

$$f(x,u) = \mu v\, Q(x,u,v).$$

If $\widehat{f}$ is a code of the (recursive, partial) $f(x,u,v)$, then

$$P(x) \iff (\forall u)[f(x,u)\downarrow]$$
$$\iff (\forall u)[\{\widehat{f}\}(x,u)\downarrow$$

$$\begin{aligned} &\iff (\forall u)[\{S_1^1(\widehat{f},x)\}(u)\downarrow] \\ &\iff S_1^1(\widehat{f},x)\in F; \end{aligned}$$

it follows that if F were Σ^0_2, then every Π^0_2 relation would be Σ^0_2, which contradicts Part (3) of the Hierarchy Theorem 5A.4.

(2) The upper bound is again obvious,

$$e\in \mathrm{Fin} \iff (\exists k)(\forall x)[x\in W_e \Longrightarrow x\leq k].$$

For the lower bound, let $P(x)$ be any Σ^0_2 relation, so that

$$P(x) \iff (\exists u)(\forall v)Q(x,u,v)$$

with some recursive Q. We set

$$g(x,u)=\mu y\,(\forall i\leq u)\neg Q(x,i,(y)_i),$$

so that if $\widehat{g}$ is a code of g, then

$$\begin{aligned} (\exists u)(\forall v)Q(x,u,v) &\iff \{u \mid g(x,u)\downarrow\} \text{ is finite} \\ &\iff \{u \mid \{\widehat{g}\}(x,u)\downarrow\} \text{ is finite} \\ &\iff \{u \mid \{S_1^1(\widehat{g},x)\}(u)\downarrow\} \text{ is finite} \\ &\iff S_1^1(\widehat{g},x)\in \mathrm{Fin}. \end{aligned}$$

This implies that Fin is not Π^0_2, because if it were, then every Σ^0_2 relation would be Π^0_2, which it is not. ⊣

The upper bound computations in this proof were trivial, and indeed they are usually very easy. When they are not completely obvious, their derivation involves applying the closure properties of Theorem 5A.3 and, in some cases, a good choice for the definition of the relation we are classifying. For example, in classifying Fin, we used without comment the equivalence

$$W_e \text{ is finite} \iff W_e \text{ is bounded};$$

we might have chosen to start with the more direct

$$W_e \text{ is finite} \iff (\exists y,n)[W_e=\{(y)_0,\ldots,(y)_n\}]$$

which is true enough but does not lead easily to the conclusion that Fin is Σ^0_2 (check it out).

Another trick which often helps, especially in the derivation of lower bounds, is to work with Π^0_k rather than Σ^0_k: i.e., to prove that $P(\vec{x})$ is Σ^0_k-complete by showing that $\neg P(\vec{x})$ is Π^0_k-complete. This, in fact, gives an easier argument for (2) of the Proposition (check it out).

Problems for Section 5A

x5A.1. Classify in the arithmetical hierarchy the set

$$A = \{e \mid W_e \subseteq \{0, 1\}\}.$$

x5A.2. Classify in the arithmetical hierarchy the set

$$B = \{e \mid W_e \text{ is finite and non-empty}\}.$$

x5A.3. Classify in the arithmetical hierarchy the set

$$C = \{x \mid \text{there exist infinitely many twin primes } \geq x\},$$

where y is a twin prime number if both y and $y + 2$ are prime.

x5A.4. Classify in the arithmetical hierarchy the relation

$$\begin{aligned} Q(e, m) &\iff \varphi_e \sqsubseteq \varphi_m \\ &\iff (\forall x)\Big(\varphi_e(x)\downarrow \implies [\varphi_m(x)\downarrow \ \&\ \varphi_e(x) = \varphi_m(x)]\Big). \end{aligned}$$

x5A.5. Classify in the arithmetical hierarchy the set

$$A = \{e \mid W_e \text{ has at least } e \text{ members}\}.$$

x5A.6. Classify in the arithmetical hierarchy the set of codes of bounded recursive partial functions,

$$B = \{e \mid \text{for some } w \text{ and all } x,\ \varphi_e(x)\downarrow \implies \varphi_e(x) \leq w\}.$$

x5A.7. Let A be any recursive set such that $A \subsetneq \mathbb{N}$; classify in the arithmetical hierarchy the set

$$B = \{e \mid W_e \subseteq A\}.$$

x5A.8. (1) Prove that the relation

$$C(e) \iff W_e \text{ is r.e. complete}$$

is arithmetical, and place this relation in some Σ^0_k or Π^0_k for as small a k as you can. (Do not try to show that your classification is complete.)

(2) Do the same for the relation

$$D(e) \iff W_e \text{ is creative.}$$

x5A.9. Prove that the graph

$$G_f(\vec{x}, w) \iff f(\vec{x}) = w$$

of a *total* function $f(\vec{x})$ is Σ^0_k if and only if it is Δ^0_k.

x5A.10*. A total function $f(\vec{x})$ is *limit recursive* if there exists a recursive, total function $g(m, \vec{x})$ such that

$$f(\vec{x}) = \lim_{m\to\infty} g(m, \vec{x}),$$

where the limit of a sequence of natural numbers is defined as usually,

$$\lim_{m\to\infty} a_m = w \iff (\exists k)(\forall m \geq k)[a_m = w].$$

Prove that a total $f(\vec{x})$ is limit recursive if and only if the graph G_f of $f(\vec{x})$ is Δ^0_2.

5B. A bit of logic

We outline here briefly the few facts that we need about **first order** or **elementary definability** on **total algebras**

$$\mathbf{M} = (M, 0, 1, f_1, \ldots, f_K) = (M, \vec{f}),$$

which we will call (first-order) **structures**. The restriction to structures simplifies things considerably, and in any case, our main interest is the classical *structure of arithmetic*

$$\mathbf{N} = (\mathbb{N}, 0, 1, +, \cdot), \tag{102}$$

The basic definitions extend naturally those in Section 2A.

5B.1. **The first-order language $\mathsf{FOL}(\mathbf{M})$: syntax.** $\mathsf{FOL}(\mathbf{M})$ is (essentially) an expansion of the language $\mathsf{T} = \mathsf{T}(\mathbf{M})$ in 2A, but we put down the definitions in full.

The alphabet of $\mathsf{FOL}(\mathbf{M})$ comprises the following symbols:

- *individual variables*: $\mathsf{v}_0, \mathsf{v}_1, \ldots$
- *individual constants*: $x(x \in M))$, including $0, 1$
- *function constants*: $\mathsf{f}_1, \ldots, \mathsf{f}_K$ $(\text{arity}(\mathsf{f}_i) = n_i)$
- the *punctuation symbols*: , ( )
- the symbol of equality: $=$
- and the *symbols of first-order logic*: $\neg$ & $\vee$ $\to$ $\exists$ $\forall$

$\mathsf{FOL}(\mathbf{M})$ differs inessentially from $\mathsf{T}(\mathbf{M})$ in that it does not have notation for branching—which makes its terms a bit simpler— but substantially because it has notation for the quantifiers $\forall$ and $\exists$.

Terms. As for $\mathsf{T}(\mathbf{M})$, these are defined by the recursion

$$A :\equiv x \mid \mathsf{v}_i \mid \mathsf{f}_i(A_1, \ldots, A_{n_i}) \tag{103}$$

And as in 2A.10 for $\mathsf{T}(\mathbf{M})$, we will rarely, if ever spell out terms correctly. In $\mathsf{FOL}(\mathbf{N})$, for example, we will write

$$x + y \text{ for } +(\mathsf{v}_1, \mathsf{v}_2),$$
$$\text{and } x(y+z)^2 \text{ for } \cdot(\mathsf{v}_3, \cdot(+(\mathsf{v}_1, \mathsf{v}_2), +(\mathsf{v}_1, \mathsf{v}_2))),$$

or sometimes

$$\mathsf{x}\,\mathsf{y} \text{ for } \cdot(\mathsf{v}_{11}, \mathsf{v}_1)$$

when it is important to make it clear that x and y are formal variables, v_{11} and v_1 in this example.

Formulas. The **prime formulas** of $\mathsf{FOL}(\mathbf{M})$ are the words of the form

$$A_1 = A_2$$

where A_1 and A_2 are terms. The **formulas** are defined recursively, starting with the prime formulas and using the operators of logic, so that every formula which is not prime is in one of the forms

$$\neg(\alpha_1) \quad (\alpha_1)\ \&\ (\alpha_2) \quad (\alpha_1) \vee (\alpha_2) \quad (\alpha_1) \to (\alpha_2) \quad \exists \mathsf{v}_i(\alpha_1) \quad \forall \mathsf{v}_i(\alpha_1)$$

where α_1, α_2 are formulas of smaller length. In compact form:

$$\alpha :\equiv A_1 = A_2 \mid \neg(\alpha_1) \mid (\alpha_1)\ \&\ (\alpha_2) \mid (\alpha_1) \vee (\alpha_2) \mid (\alpha_1) \to (\alpha_2) \mid \exists \mathsf{v}_i(\alpha_1) \mid \forall \mathsf{v}_i(\alpha_1) \tag{104}$$

We will also use simplified forms for formulas, writing for example

$$(\forall \mathsf{x})(\forall \mathsf{y})[\mathsf{x} + \mathsf{y} = \mathsf{y} + \mathsf{x}] \text{ for } \forall \mathsf{v}_1(\forall \mathsf{v}_2(+(\mathsf{v}_1, \mathsf{v}_2) = +(\mathsf{v}_2, \mathsf{v}_1))) \tag{$*$}$$

A formula α is **pure** if no constants from M occur in it except (perhaps) $0, 1$.

The terms and formulas of $\mathsf{FOL}(\mathbf{M})$ satisfy *unique readability* lemmas like Lemma 2A.6 for T. These are quite routine to formulate and prove (by induction) and we will not take them up in detail. They are, however, important, as they allow us to prove properties of terms and formulas *by induction* and to define operations on them *by recursion*, as we did for the terms of $\mathsf{T}(\mathbf{M})$ by appealing to Lemma 2A.7.

We will define precisely the semantics of $\mathsf{FOL}(\mathbf{M})$ below, but intuitively, a formula α means what its natural translation into (mathematical) English says: the formula in $(*)$, for example, is read

"for all x and all y, $x + y = y + x$",

and it expresses in $\mathsf{FOL}(\mathbf{N})$ the commutative law for addition. However, the quantifiers $\exists$ and $\forall$ bring to the language some new phenomena that do not show up in the simpler $\mathsf{T}(\mathbf{M})$. It is useful to look at some examples before we go into the formal definitions.

Consider the following four formulas in the language of arithmetic and their meaning in the structure **N**:

$$\alpha \equiv \mathsf{x}\,\mathsf{y} = \mathsf{y}, \quad \beta \equiv (\forall \mathsf{x})[\mathsf{x}\,\mathsf{y} = \mathsf{y}], \quad \gamma \equiv (\forall \mathsf{x})(\forall \mathsf{y})[\mathsf{x}\,\mathsf{y} = \mathsf{y}], \quad \delta \equiv \gamma \vee \mathsf{x} = 0.$$

Whether α is true or not evidently depends on what the variables x, y are set to: if $\mathsf{x} := x$ and $\mathsf{y} := y$, then, clearly,

$$\mathsf{x}\,\mathsf{y} = \mathsf{y} \text{ is true } \iff xy = y \iff y = 0 \text{ or } x = 1.$$

The truth value of the second formula β is independent of what x stands for: if $\mathsf{y} := y$, then

$$(\forall \mathsf{x})[\mathsf{x}\,\mathsf{y} = \mathsf{y}] \text{ is true } \iff \text{for all } x,\ xy = y \iff y = 0.$$

The third formula γ does not depend on the values of x and y:

$$(\forall \mathsf{x})(\forall \mathsf{y})[\mathsf{x}\,\mathsf{y} = \mathsf{y}] \text{ is true } \iff \text{for all } x \text{ and } y,\ xy = y \iff \text{ff},$$

i.e., γ is plain false. The fourth one is more interesting; if $\mathsf{x} := x$, then

$$(\forall \mathsf{x})(\forall \mathsf{y})[\mathsf{x}\,\mathsf{y} = \mathsf{y}] \vee \mathsf{x} = 0 \text{ is true } \iff \text{ff or } x = 0 \iff x = 0.$$

The interesting bit is that although the variable x has three occurrences in this formula, the first two "don't count", and we can replace x by any $\mathsf{z} \not\equiv \mathsf{y}$ in them without changing the formula's meaning,

$$(\forall \mathsf{x})(\forall \mathsf{y})[\mathsf{x}\,\mathsf{y} = \mathsf{y}] \vee \mathsf{x} = 0 \text{ is true } \iff (\forall \mathsf{z})(\forall \mathsf{y})[\mathsf{z}\,\mathsf{y} = \mathsf{y}] \vee \mathsf{x} = 0 \text{ is true.}$$

The value of x comes into play in its third occurrence and determines the truth value of the formula. We say that "x is *bound* in its first two occurrences but *free* in its third", by the following precise definition of these notions:

5B.2. **Free and bound occurrences of variables.** The *free* occurrences of variables in formulas are defined by recursion as follows.

(F1) $\mathrm{FO}(A = B) =$ all the occurrences of variables in $A = B$.
(F2) $\mathrm{FO}(\neg(\alpha)) = \mathrm{FO}(\alpha)$, $\mathrm{FO}((\alpha)\ \&\ (\beta)) = \mathrm{FO}(\alpha) \cup \mathrm{FO}(\beta)$, and similarly for the other connectives.
(F3) $\mathrm{FO}(\forall \mathsf{v}_i(\alpha)) = \mathrm{FO}(\exists \mathsf{v}_i(\alpha)) = \mathrm{FO}(\alpha) \setminus \{\mathsf{v}_i\}$, meaning that we remove from the free occurrences of variables in α all occurrences of v_i.

In particular, the first occurrence of v_i in $\exists \mathsf{v}_i(\alpha)$ is not free.

An occurrence of a variable which is not free in a formula α is *bound* in α. The *free variables* of α are the variables which have at least one free occurrence in α and, intuitively, the truth of falsity of α depends only on the values assigned to them.

A term or formula is **closed** if it has no free variables; and a closed formula is a **sentence**.

5B.3. **The free substitution operation**. For any formula α, individual variable v and $x \in M$, we set

(105) $\alpha\{\mathsf{v} :\equiv x\}$ = the result of replacing v by x in all the free occurrences of v in α.

For example, $\Big(\forall \mathsf{u}[\mathsf{u}+\mathsf{v} = \mathsf{u}]\Big)\{\mathsf{v} :\equiv 12\} \equiv \forall \mathsf{u}[\mathsf{u}+12 = \mathsf{u}]$, and the restriction to replacing only into free occurrences of variables is needed because, in this example, $\Big(\forall \mathsf{u}[\mathsf{u}+\mathsf{v} = \mathsf{u}]\Big)\{\mathsf{u} :\equiv 12\} \equiv (\forall 12[12+\mathsf{v} = 12]\Big)$ which is evidently meaningless.

5B.4. **The first-order language $\mathsf{FOL} = \mathsf{FOL}(\mathbf{M})$: semantics.** We now extend to the sentences of $\mathsf{FOL}(\mathbf{M})$ the denotation function $\mathbf{val}_{\mathbf{M}}(A)$ defined on closed terms in 2A.9. The denotation of a sentence is a *truth value* tt or ff, and here it is really easier to use model theoretic notation: we will define the **satisfaction** or **truth relation**

$$\mathbf{M} \models \alpha \iff \mathbf{M} \text{ satisfies } \alpha \iff \alpha \text{ is true in } \mathbf{M}$$

and then set

$$\mathbf{val}_{\mathbf{M}}(\alpha) = \begin{cases} \text{tt}, & \text{if } \mathbf{M} \models \alpha, \\ \text{ff}, & \text{otherwise.} \end{cases}$$

The definition is by recursion on the sentence α:

$$\begin{aligned}
\mathbf{M} \models A = B &\iff \mathbf{val}_{\mathbf{M}}(A) = \mathbf{val}_{\mathbf{M}}(B) \\
\mathbf{M} \models \neg(\alpha_1) &\iff \mathbf{M} \not\models \alpha_1 \ (\iff \text{ it is not the case that } \mathbf{M} \models \alpha_1) \\
\mathbf{M} \models (\alpha_1)\ \&\ (\alpha_2) &\iff \mathbf{M} \models \alpha_1 \text{ and } \mathbf{M} \models \alpha_2 \\
\mathbf{M} \models (\alpha_1) \vee (\alpha_2) &\iff \mathbf{M} \models \alpha_1 \text{ or } \mathbf{M} \models \alpha_2 \\
\mathbf{M} \models (\alpha_1) \to (\alpha_2) &\iff \mathbf{M} \not\models \alpha_1 \text{ or } \mathbf{M} \models \alpha_2 \\
\mathbf{M} \models \exists \mathsf{v}_i(\alpha_1) &\iff \text{for some } x \in M,\ \mathbf{M} \models \alpha_1\{\mathsf{v} :\equiv x\} \\
\mathbf{M} \models \forall \mathsf{v}_i(\alpha_1) &\iff \text{for every } x \in M, \mathbf{M} \models \alpha_1\{\mathsf{v} :\equiv x\}.
\end{aligned}$$

These famous conditions are due to Tarski, who was criticized for "claiming the obvious" since all they do is to explain how to read the formal expressions of FOL in English. Indeed, this is all they do—and it is what they should do; but they are very useful, nonetheless, because they allow us to make precise and prove rigorously by induction many properties of formulas which are not easy to formulate precisely or check easily from their translation into English. One of the most basic of these is the following:

5B.5. **Elementary relations and functions.** Suppose $\mathbf{M}$ is a structure; α is a *pure formula*, i.e., no constants from M occur in α except perhaps $0, 1$; $\vec{\mathsf{x}} \equiv \mathsf{x}_1, \dots, \mathsf{x}_n$ is a list of distinct variables which includes all

the free variables of α; and $R(x_1, \dots, x_n)$ is an n-ary relation on M. We say that α *defines* R *in* $\mathbf{M}$ *with respect to* $\vec{\mathsf{x}}$, if

$$R(\vec{x}) \iff \mathbf{M} \models \alpha\{\mathsf{x}_1 :\equiv x_1, \dots, \mathsf{x}_n :\equiv x_n\} \quad (\vec{x} \in M^n). \tag{106}$$

A relation $R(\vec{x})$ is **first-order definable** in $\mathbf{M}$ or **elementary** in $\mathbf{M}$ or just **M-elementary** if it is defined by a pure formula with respect to some list of distinct variables; and a function $f : M^n \to M$ is **elementary** if its graph (85) is an elementary relation.

Problems for Section 5B

x5B.1. For each occurrence of a variable in the following formulas, determine whether it is free or bound:

$$\alpha \equiv \mathsf{f}_1(\mathsf{v}_1, 0) = \mathsf{f}_2(\mathsf{v}_2), \quad \beta \equiv \exists \mathsf{v}_1(\mathsf{f}_1(\mathsf{v}_1, 0) = \mathsf{f}_2(\mathsf{v}_2))$$
$$\gamma \equiv (\mathsf{f}_1(\mathsf{v}_1, 0) = \mathsf{f}_2(\mathsf{v}_2)) \ \&\ (\exists \mathsf{v}_1(\mathsf{f}_1(\mathsf{v}_1, 0) = \mathsf{f}_2(\mathsf{v}_2))), \quad \delta :\equiv \forall \mathsf{v}_2(\gamma),$$

x5B.2 (Renaming of variables). Let $\mathbf{M}$ be any structure and suppose that α defines $R(\vec{x})$ in $\mathbf{M}$ with respect to the list of distinct variable $\mathsf{x}_1, \dots, \mathsf{x}_n$, i.e., (106) holds. Let $\mathsf{y}_1, \dots, \mathsf{y}_n$ be any list of distinct variables which do not occur (free or bound) in α, and let

$$\beta :\equiv \alpha\{\mathsf{x}_1 :\equiv \mathsf{y}_1, \dots, \mathsf{x}_n :\equiv \mathsf{y}_n\}$$

be the result of replacing every x_i by y_i in α. Prove that β defines $R(\vec{x})$ in $\mathbf{M}$ with respect to $\mathsf{y}_1, \dots, \mathsf{y}_n$, i.e.,

$$R(\vec{x}) \iff \mathbf{M}, \{\mathsf{y}_1 := x_1, \dots, \mathsf{y}_n := x_n\} \models \beta, \quad (\vec{x} \in M^n). \tag{107}$$

Infer that if $P(\vec{x})$ and $Q(\vec{x})$ are both first-order definable in a structure $\mathbf{M}$, then there exist pure formulas α and β and a list $\mathsf{z}_1, \dots, \mathsf{z}_n$ of distinct variables such that α defines $P(\vec{x})$ in $\mathbf{M}$ with respect to $\mathsf{z}_1, \dots, \mathsf{z}_n$ and β defines $Q(\vec{x})$, also with respect to $\mathsf{z}_1, \dots, \mathsf{z}_n$. HINT: Use induction on α.

This simple fact is very useful in proving the closure properties of the class of relations which are elementary in a structure $\mathbf{M}$. For example:

x5B.3. (1) Prove that if $P(\vec{x})$ and $Q(\vec{x})$ are both elementary in $\mathbf{M}$, then so is their conjunction

$$R(\vec{x}) \iff P(\vec{x}) \ \&\ Q(\vec{x}).$$

(2) Prove that if $P(\vec{x}, y)$ is elementary in $\mathbf{M}$, then so is

$$R(\vec{x}) \iff (\exists y) P(\vec{x}, y).$$

5C. Arithmetical relations and functions

The study of first-order definability in arbitrary structures is one of the main aims of **model theory** and a tremendous amount of work has gone—and is going—into it. Here we are concerned only with the elementary relations in the standard structure **N** of arithmetic, and our first aim is to show that they coincide with the **arithmetical relations** on $\mathbb{N}$, those which occur in the arithmetical hierarchy defined in Section 5A.

5C.1. Theorem. *The class $\mathcal{A}$ of elementary relations in* **N** *is the smallest class of relations on $\mathbb{N}$ with the following properties*:

(1) *The relations*

$$x = y, \quad x = 0, \quad x = 1, \quad x + y = z, \quad x \cdot y = z$$

are in $\mathcal{A}$.

(2) *$\mathcal{A}$ is closed under substitutions of projections $P_i^n(\vec{x})$ and under the logical operators, $\neg$, & , $\vee$, $\to$, $\exists$, $\forall$.*

It follows that the inequality relation

$$x \leq y \iff (\exists z)[x + z = y]$$

is elementary in **N**, *and that the class of* **N**-*elementary relations is closed under* **N**-*elementary substitutions, since*

$$\begin{aligned} &P(f_1(\vec{x}), \dots, f_m(\vec{x})) \\ &\iff (\exists w_1) \cdots (\exists w_m)[f_1(\vec{x}) = w_1 \ \& \ \cdots \ \& \ f_m(\vec{x}) = w_m \ \& \ P(w_1, \dots, w_m)]. \end{aligned}$$

Outline of proof. Let $\mathcal{B}$ be any collection of relations on $\mathbb{N}$ which satisfies (1) and (2) and prove by induction on the formulas that if α defines $R(\vec{x})$ with respect to some $\mathsf{x}_1, \dots, \mathsf{x}_n$, then $R(\vec{x})$ is in $\mathcal{B}$. The argument requires repeated appeals to Problem x5B.2 but is quite easy. It proves that every relation which is elementary in **N** belongs to every class of relations $\mathcal{B}$ which satisfies (1) and (2). For the converse, it is enough to prove that $\mathcal{A}$ satisfies (1) and (2), and this is also easy, by (essentially) the same argument. ⊣

This simple theorem seems to suggest that the language of arithmetic cannot deal with exponentiation, which would make it a very poor language indeed. In fact 2^x is **N**-elementary, as are all primitive recursive functions and relations. We need the following standard result from number theory to prove this.

5C.2. The Chinese Remainder Theorem. *Suppose that $d_0, \dots, d_t$ are relatively prime numbers, i.e., no two of them have a common factor other*

than 1, *and* $w_0 < d_0, \dots, w_t < d_t$. *Then there exists some number* a *such that*

$$w_0 = \mathrm{rem}(a, d_0), \dots, w_t = \mathrm{rem}(a, d_t).$$

Proof. Consider the set D of all $(t+1)$-tuples bounded by the given numbers $d_0, \dots, d_t$,

$$D = \{(w_0, \dots, w_t) \mid w_0 < d_0, \dots, w_t < d_t\}.$$

This has $|D| = d_0 d_1 \cdots d_t$ members. Let

$$A = \{a \mid a < |D|\}$$

which also has $d_0 d_1 \cdots d_t$ members, and define the function $f : A \to D$ by

$$f(a) = (\mathrm{rem}(a, d_0), \mathrm{rem}(a, d_1), \dots, \mathrm{rem}(a, d_t)).$$

Notice that f is one-to-one, because if $f(a) = f(b)$ with $a < b < |D|$, then $b - a$ is divisible by each of $d_0, \dots, d_t$ and hence by their product $|D|$ by Problem x5C.2; so $|D| \leq b - a$, which is absurd since $a < b < |D|$. We now apply the Pigeonhole Principle: since A and D are equinumerous and $f : A \rightarrowtail D$ is an injection, it must be a surjection, and hence whatever the tuple $(w_0, \dots, w_t)$, there is an $a < d$ such that

$$\pi(a) = (\mathrm{rem}(a, d_0), \mathrm{rem}(a, d_1), \dots, \mathrm{rem}(a, d_t)) = (w_0, \dots, w_t). \qquad \dashv$$

5C.3. Lemma (Gödel's β-function). *The function*

$$\beta(a, b, i) = \mathrm{rem}(a, 1 + (i+1)b)$$

is **N**-*elementary, and for every sequence of numbers* $w_0, \dots, w_y$, *there exist natural numbers* a *and* b, *such that*

$$w_i = \beta(a, b, i) \qquad (i = 0, \dots, y).$$

Proof. The function $\beta(a, b, i)$ is elementary in **N** because

$$\beta(a, b, i) = w \iff (\exists c)\Big(a = (1 + (i+1)b)c + w \;\&\; w < 1 + (i+1)b\Big).$$

For the second claim, let

$$\begin{aligned} d &= \max(w_0, \dots, w_y, y) + 1 \\ b &= d\,! \\ z_i &= 1 + (i+1)b = 1 + (i+1)d\,! \quad (i = 0, \dots, y). \end{aligned}$$

Note that the numbers $z_0, z_1, \dots, z_y$ are *relatively prime*, i.e., there is no prime number dividing any two of them; because if p is prime and a common divisor of z_i and z_j with $i < j \leq y$, then:

1. $p > d$, otherwise $p | d\,!$, and this is absurd, since $p | 1 + (i+1)d\,!$, and
2. p divides the difference $(j - i)d\,!$, so $p | (j - i)$ or $p | d\,!$,

and this is also absurd, since $j-i \leq y < d < p$ and p does not divide $d!$, as in 1. Finally, by the Chinese Remainder Theorem, since $w_0 < z_0, \dots, w_y < z_y$, there exists some a such that

$$w_0 = \mathrm{rem}(a, z_0) = \beta(a, b, 0), \dots, w_y = \mathrm{rem}(a, z_y) = \beta(a, b, y). \qquad \dashv$$

5C.4. Theorem. (1) *Every primitive recursive function is elementary in* $\mathbf{N}$.

(2) *A relation is elementary in* $\mathbf{N}$ *if and only if it is arithmetical, i.e., it occurs in* Σ^0_k *for some* k. *In symbols:*

$$\mathcal{A} = \bigcup_k \Sigma^0_k = \bigcup_k \Pi^0_k.$$

Proof. (1) It suffices to show that the functions S, C^n_q and P^n_i are $\mathbf{N}$-elementary, and that the set of $\mathbf{N}$-elementary functions is closed under composition and primitive recursion, and from these only the last is not trivial.

If the function $f(y, \vec{x})$ is defined by the primitive recursion

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

then, by Dedekind's analysis of recursion (34),

$$\begin{aligned} f(y, \vec{x}) = w \iff (\exists w_0, \dots, w_y)[g(\vec{x}) = w_0 \ \&\ w = w_y \\ \&\ (\forall i < y)[h(w_i, i, \vec{x}) = w_{i+1}]]; \end{aligned}$$

this is obvious, with $w_i = f(i, \vec{x})$ for the direction ($\Rightarrow$), and by induction on $i \leq y$ for the direction ($\Leftarrow$). It follows, by the Lemma, that

$$\begin{aligned} f(y, \vec{x}) = w \iff (\exists a)(\exists b)[g(\vec{x}) = \beta(a, b, 0) \ \&\ w = \beta(a, b, y) \\ (\forall i < y)[h(\beta(a, b, i), i, \vec{x}) = \beta(a, b, i+1)]] \end{aligned}$$

which implies immediately, by the closure properties of $\mathcal{A}$, that the graph of $f(y, \vec{x})$ is in $\mathcal{A}$.

(2) The inclusion $\Sigma^0_k \subseteq \mathcal{A}$ is simple, by induction on k, and the inverse inclusion $\mathcal{A} \subseteq \bigcup_k \Sigma^0_k$ follows from the characterization 5C.1. $\dashv$

Problems for Section 5C

x5C.1 (Exponentiation is not polynomial). Prove that the exponential 2^x is not defined by a term in $\mathbf{N}$: i.e., there is no term A of the language of arithmetic with just one variable x such that for every $x \in \mathbb{N}$,

$$\mathbf{val}_{\mathbf{N}}(A\{\mathsf{x} :\equiv x\}) = 2^x.$$

x5C.2. Prove that it $d_1, \dots, d_n$ are relatively prime and each of them divides a number d, then their product $d_1 d_2 \cdots d_n$ divides d. HINT: There are many ways to prove this basic fact. Perhaps the simplest way to see it is to appeal to the *Fundamental Theorem of arithmetic*: *Every number $w > 1$ can be expressed uniquely as a product $w = q_1 \cdots q_m$ of a sequence of primes which is non-decreasing, $q_1 \leq \cdots \leq q_m$.*

5D. The theorems of Tarski, Gödel and Church

To prove these basic results, we will use once more the method of *coding* (or *arithmetization*), characteristic of the subject.

5D.1. **Coding of pure formulas.** With each symbol c of the first-order language of arithmetic $\mathsf{FOL} = \mathsf{FOL}(\mathbf{N})$ (other than constants > 1), we associate a natural number $[c]$ by the simple enumeration,

0	1	$+$	$\cdot$	$=$	$\neg$	$\&$	$\vee$	$\rightarrow$	$\exists$	$\forall$	$($	$)$	$,$	v_0	v_1	$\dots$
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	$\dots$

so that $[\exists] = 9$, $[\mathsf{v}_0] = 14$, etc.; and we code words from this alphabet as usual, using sequence codes,

$$[s_0 s_1 \cdots s_n] = \langle [s_0], [s_1], \dots, [s_n] \rangle$$

e.g.,

$$[\exists \mathsf{v}_2 (\mathsf{0} = \mathsf{v}_2)] = \langle [\exists], [\mathsf{v}_2], [(], [\mathsf{0}], [=], [\mathsf{v}_2], [)] \rangle.$$

By the methods of Chapter 3, it is not difficult to show that the basic *metamathematical*, syntactical relations of the language are primitive recursive in the codes. These include

$$\begin{aligned} \text{Formula}(a) &\iff a \text{ is the code of a (pure) formula} \\ \text{Free}(a, i) &\iff a \text{ is the code of a formula } \alpha \\ &\qquad\qquad \text{and } \mathsf{v}_i \text{ occurs free in } \alpha \\ \text{Sentence}(f) &\iff a \text{ is the code of a sentence,} \end{aligned}$$

etc. Finally we set

$$\begin{aligned} (108) \quad \text{Truth} &= \text{Truth}(\mathbf{N}) \\ &= \{a \mid a \text{ is the code of a pure sentence which is true in } \mathbf{N}\}. \end{aligned}$$

This is the set of natural numbers which codifies all (first-order) **arithmetical truths**: every interesting proposition of number theory can be expressed by a pure sentence of $\mathsf{FOL}(\mathbf{N})$, and so it is true exactly when its code is in Truth.

5D.2. Lemma. *Every arithmetical relation $R(\vec{x})$ is* 1-1 *reducible to the set* Truth*, i.e., there exists a* (one-to-one) *recursive function $f(\vec{x})$ such that*

$$R(\vec{x}) \iff f(\vec{x}) \in \text{Truth}. \tag{109}$$

Proof. For each natural number x we define the term $\Delta(x)$ by the recursion,

$$\Delta(0) \equiv 0, \quad \Delta(x+1) \equiv (\Delta(x)) + (1),$$

so that the **numeral** $\Delta(x)$ is closed and denotes (in $\mathbf{N}$) the number x,

$$\mathbf{val}(\Delta(x)) = x.$$

It follows (easily, by the Tarski conditions) that for every formula α with free variables in the list $\mathsf{x}_1, \ldots, \mathsf{x}_n$,

$$\begin{aligned}
&\mathbf{N} \models \alpha\{\mathsf{x}_1 :\equiv x_1, \ldots, \mathsf{x}_n :\equiv x_n\} \\
&\quad \iff \mathbf{N} \models (\exists \mathsf{x}_1)\cdots(\exists \mathsf{x}_n)[\mathsf{x}_1 = \Delta(x_1) \ \&\ \cdots \ \&\ \mathsf{x}_n = \Delta(x_n) \ \&\ \alpha] \\
&\quad \iff [(\exists \mathsf{x}_1)\cdots(\exists \mathsf{x}_n)[\mathsf{x}_1 = \Delta(x_1) \ \&\ \cdots \ \&\ \mathsf{x}_n = \Delta(x_n) \ \&\ \alpha]] \in \text{Truth};
\end{aligned}$$

and so, if the formula α defines the relation $R(\vec{x})$ as in (106), then (109) holds with the function

$$f(\vec{x}) = [(\exists \mathsf{x}_1)\cdots(\exists \mathsf{x}_n)[\mathsf{x}_1 = \Delta(x_1) \ \&\ \cdots \ \&\ \mathsf{x}_n = \Delta(x_n) \ \&\ \alpha]]$$

which is easily recursive and one-to-one. ⊣

5D.3. **Tarski's Theorem.** *The set* Truth *is not arithmetical, and in particular, it is not recursive.*

Proof. If Truth were arithmetical, then it would be Σ^0_k, for some k, and so, by the Lemma, every arithmetical relation would be Σ^0_k, which contradicts the Hierarchy Theorem 5A.4 (3). ⊣

It is hard to overemphasize the foundational significance of Tarski's Theorem. Combined with the Church-Turing Thesis, it says in part that *there is no algorithm which can decide whether an arbitrary sentence of arithmetic is true or false*—even if we just want to know the truth value, i.e., we do not ask for an algorithm which produces *justifications* for the true sentences.

Mathematicians, however, are very much interested in justifications, i.e., *proofs*. Gödel's celebrated *First Incompleteness Theorem* deals with this aspect of the foundations of arithmetic and places some severe restrictions on proof systems for arithmetic, what we might call attempts to *axiomatize* the theory of numbers. We cannot do it full justice here, since we have not studied the relevant material from logic; but we can formulate and prove an abstract result which captures its essential content.

5D.4. **Proof systems.** A *proof system* for arithmetic is any set $\mathcal{P}$ of finite sequences $(\alpha_0, \dots, \alpha_n)$ of pure $\mathsf{FOL}(\mathbf{N})$-formulas such that the last formula α_n is a sentence. The *proofs* of $\mathcal{P}$ are its members, and we write

$$\vdash_{\mathcal{P}} \alpha \iff \text{there exists a proof } (\alpha_0, \dots, \alpha_n) \in \mathcal{P} \text{ with } \alpha_n \equiv \alpha.$$

If $\vdash_{\mathcal{P}} \alpha$, we say that α is a *theorem of* $\mathcal{P}$.

In studies of formal, axiomatic number theory, we consider specific proof systems which attempt to capture our intuitions about logic and arithmetic. Typically we call a sequence $(\alpha_0, \dots, \alpha_n)$ a *formal proof* if each α_i is either an *axiom of logic* or an *axiom of arithmetic*, or it follows by some logical or arithmetical *rule of inference* from formulas $\alpha_{j_1}, \dots, \alpha_{j_l}$ with $j_1, \dots, j_l < i$ listed earlier in the proof than α_i. Different choices of axioms and rules of inference lead to different proof systems that express different intuitions, but always have some claim that the proofs that are accepted provide justifications for the theorems.

This preliminary notion allows for very silly proof systems, of course, e.g., the set of all sequences of formulas which end with a sentence, or the set of all sequences of length 1 whose only member is a true sentence α, etc. There are, however, two obvious and reasonable conditions we can impose on a proof system $\mathcal{P}$ which make it worth considering.

(1) **Soundness**: *For every pure sentence* α*,*

$$\textit{if } \vdash_{\mathcal{P}} \alpha, \textit{ then } \mathbf{N} \models \alpha,$$

so that $\mathcal{P}$ *proves only true sentences.*

(2) **Decidability for proofs**: *The relation*

$$\mathrm{Proof}_{\mathcal{P}}(y) \iff (\exists \alpha_0, \dots, \alpha_n \in \mathcal{P})[y = \langle [\alpha_0], \dots, [\alpha_n] \rangle]$$

is recursive.

The second condition expresses (by the Church-Turing Thesis) the basic practice of mathematics, that Somebody's claim that he has proved some theorem (the existence of infinitely many twin primes, for example), can be "checked"—there is a generally accepted effective method which decides whether Somebody's ramblings constitutes a rigorous proof of his claim or not, according to the axioms and the rules of logic that we have accepted.

All specific proof systems which are seriously studied are decidable for proofs and sound—or, at the least, are hoped to be sound by their inventors.

There is a third property which we would want a good proof system $\mathcal{P}$ to have:

(3) **Completeness**: *For every pure sentence* α*,*

$$\text{either } \vdash_{\mathcal{P}} \alpha \text{ or } \vdash_{\mathcal{P}} \neg\alpha.$$

A proof system for arithmetic which would be sound, decidable for proofs and complete, would "codify" in some specific way some basic principles of logic and arithmetic which suffice to solve all problems of number theory. There were many serious attempts in the first third of the 20th century to find such a system, but they all failed—and they could not have succeeded:

5D.5. **Gödel's 1st Incompleteness Theorem.** *There is no proof system for arithmetic which is sound, decidable for proofs and complete.*

Proof. If $\mathcal{P}$ has all three properties, then for each pure sentence α

$$\vdash_{\mathcal{P}} \alpha \Longrightarrow \mathbf{N} \models \alpha$$

because the system is sound, and

$$\begin{aligned} \mathbf{N} \models \alpha &\implies \mathbf{N} \not\models \neg\alpha \\ &\implies \not\vdash_{\mathcal{P}} \neg\alpha \quad \text{(by soundness)} \\ &\implies \vdash_{\mathcal{P}} \alpha \quad \text{(by completeness).} \end{aligned}$$

It follows that

$$\begin{aligned} \text{Truth} &= \{[\alpha] \mid \vdash_{\mathcal{P}} \alpha\} \\ &= \{e \mid \text{Sentence}(e) \ \&\ (\exists y)[\text{Proof}_{\mathcal{P}}(y) \ \&\ e = \text{last}(y)]\}, \end{aligned}$$

and so the set Truth is Σ^0_1, by the hypothesis of decidability for proofs for $\mathcal{P}$. This contradicts Tarski's Theorem 5D.3. ⊣

Whether the Church-Turing Thesis is true or not does not come up in the specific applications of the 1st Incompleteness Theorem, because in practice, the axiomatic systems that have been studied are all decidable for proofs. It is hard to imagine a useful axiomatization of number theory where we could not tell whether an alleged proof is indeed acceptable.

To formulate the last basic result of this section, consider the first-order language $\mathsf{FOL}(\mathbf{M})$ for an arbitrary structure $\mathbf{M} = (M, 0, 1, f_1, \dots, f_K)$. A pure sentence α of $\mathsf{FOL}(\mathbf{M})$ can be interpreted in every structure

$$\mathbf{M}' = M, 0', 1', f'_1, \dots, f'_K)$$

of the same vocabulary $\boldsymbol{v} = (\mathsf{f}_1, \dots, \mathsf{f}_K)$ and for such sentences we set

$$\models \alpha \iff \text{for every algebra } \mathbf{M}' \text{ with } \boldsymbol{v}(\mathbf{M}') = \boldsymbol{v}(\mathbf{M}),\ \mathbf{M}' \models \alpha.$$

These are the **valid pure sentences** of $\mathsf{FOL}(\mathbf{M})$, those which are true in every structure, independently of the universe M or the interpretations $f_1, \dots, f_K$ of the constant symbols. They include trivial sentences like

$$\alpha \vee \neg\alpha,$$

and one might guess that they are all trivial, but this is not true.

By the classical *Completeness Theorem* of Gödel,

$$\models \alpha \iff \alpha \text{ is a theorem of classical, first-order logic,}$$

and the decidability for proofs of classical logic implies that *for every vocabulary* $(\mathsf{f}_1, \dots, \mathsf{f}_K)$*, the set of* (codes of) *valid pure sentences of* $\mathsf{FOL}(\mathbf{M})$ *is semirecursive.* On the other hand:

5D.6. **Church's Theorem**. *For some structure* $\mathbf{M}$*, the problem whether a pure sentence* α *of* $\mathsf{FOL}(\mathbf{M})$ *is valid is unsolvable.*

Outline of proof. Let

$$E = (f_0, f_1, \dots, f_K)$$

be any primitive recursive program according to definition 1B.17. It is not difficult to construct a first order sentence

$$\alpha_E = \alpha_0 \ \& \ \cdots \ \& \ \alpha_K$$

using formal symbols $0, 1, \mathsf{S}, \mathsf{Pd}, \mathsf{f}_0, \dots, \mathsf{f}_K$ which "expresses formally" the definitions in E; if, e.g., the projection symbol P^3_2 occurs in E, then the sentence

$$(\forall \mathsf{v}_1)(\forall \mathsf{v}_2)(\forall \mathsf{v}_3)[\mathsf{P}^3_2(\mathsf{v}_1, \mathsf{v}_2.\mathsf{v}_3) = \mathsf{v}_2]$$

is one of the α_i's, and if f is defined in E by the primitive recursion

$$f(0) = 5$$
$$f(y+1) = h(f(y), y),$$

then the sentence

$$\mathsf{f}(0) = \Delta(5) \ \& \ (\forall \mathsf{v}_1)[\mathsf{f}(\mathsf{S}(\mathsf{v}_1)) = \mathsf{h}(\mathsf{f}(\mathsf{v}_1), \mathsf{v}_1)]$$

is also one of the α_i's. With this definition, it follows without great difficulty that for each function f_i defined in E and all $\vec{x} = x_1, \dots, x_n, w \in \mathbb{N}$,

$$f_i(\vec{x}) = w \iff \models \alpha_E \to \mathsf{f}_i(\Delta(x_1), \dots, \Delta(x_n)) = \Delta(w).$$

Finally, we apply this Lemma to the case that f_K is the characteristic function of the relation $T_1(x, x, y)$ and we show quite easily that the decidability of the validity relation leads to a contradiction. ⊣

This is (essentially) Church's proof, but the result was also independently proved by Turing. It hence since been extended to several specific vocabularies, including that of arithmetic: i.e., *the set of all pure, valid* $\mathsf{FOL}(\mathbf{N})$ *sentences is undecidable.*

The decision problem for provability in first order logic was a big open question at the time (1936), heavily promoted by none other than David Hilbert and generally referred to in German as the *Entscheidungsproblem.* Its solution did much to popularize the Church-Turing Thesis and the then emerging theory of computability among mathematicians.

CHAPTER 6

RECURSIVE FUNCTIONALS AND EFFECTIVE OPERATIONS

In this chapter we will study generalized programs, "non-deterministic" and with "external" function variables, and we will explore their applications, especially in the theory of "computable functionals". Most of the results we will prove hold for all partial algebras, but the most interesting phenomena manifest themselves fully in the classical algebra $\mathbf{N}_0 = (\mathbb{N}, 0, 1, S, Pd)$ and its expansions.

6A. Recursive functionals

6A.1. DEFINITION. A **generalized recursive program** of $\mathbf{N}_0$ is any system of recursive equations

$$(E)\qquad \begin{aligned} (e_0)\quad & \mathsf{p}_0(\vec{x}_0) = E_0 = E_0[\vec{x}_0, \mathsf{p}_0, \dots, \mathsf{p}_k, \mathsf{q}_1, \dots, \mathsf{q}_l] \\ & \vdots \\ (e_k)\quad & \mathsf{p}_k(\vec{x}_k) = E_k = E_k[\vec{x}_k, \mathsf{p}_0, \dots, \mathsf{p}_k, \mathsf{q}_1, \dots, \mathsf{q}_l] \end{aligned}$$

where the function variables $\mathsf{p}_0, \dots, \mathsf{p}_k, \mathsf{q}_1, \dots, \mathsf{q}_l$ are distinct, as in the basic definition in section 2B, but E supplies definitions only for the **internal** (bound) variables $\mathsf{p}_0, \dots, \mathsf{p}_k$ while it also allows the occurrence of **external** (free) variables $\mathsf{q}_1, \dots, \mathsf{q}_l$ in the pure terms $E_1, \dots, E_k$, as the notation $E_i[\vec{x}_i \mathsf{p}_0, \dots, \mathsf{p}_k, \mathsf{q}_1, \dots, \mathsf{q}_l]$ suggests. A program of this kind is interpreted naturally in arbitrary expansions

$$(\mathbf{N}_0, q_1, \dots, q_l) = (\mathbb{N}, 0, 1, S, Pd, q_1, \dots, q_l)$$

of $\mathbf{N}_0$, i.e., if we consider the variables $\mathsf{q}_1, \dots, \mathsf{q}_l$ as constants which name given partial functions $q_1, \dots, q_l$, just as $\mathsf{f}_1, \dots, \mathsf{f}_K$ name the given partial functions $f_1, \dots, f_K$ of a partial algebra. With the notation of (60), we set

$$\begin{aligned} (110)\quad & \alpha_E(\vec{x}, q_1, \dots, q_l) = w \\ & \iff (\mathbf{N}_0, q_1, \dots, q_l), E \vdash \mathsf{p}_0(\vec{x}) = w \text{ with } \mathsf{q}_1 := q_1, \dots, \mathsf{q}_l := q_l \\ & \iff E, \vec{q} := \vec{\mathsf{q}} \vdash \mathsf{p}_0(\vec{x}) = w. \end{aligned}$$

More generally, for each internal variable p of E,

$$(111)\quad E, \vec{\mathsf{q}} := \vec{q} \vdash \mathsf{p}(\vec{x}) = w \iff (\mathbf{N}_0, q_1, \dots, q_l), E \vdash \mathsf{p}(\vec{x}) = w \text{ with } \mathsf{q}_1 := q_1, \dots, \mathsf{q}_l := q_l.$$

The notation here omits the reference to the algebra $\mathbf{N}_0$ since this is the only algebra which we will use in this chapter. It shows explicitly the **assignment** $\mathsf{q}_1 := q_1, \dots, \mathsf{q}_l := q_l$ to the external variables: this is useful, as we will often interpret the same program with different–all possible—assignments to its external variables.

The programs we studied in Chapter 2 (without external function variables) are now called **autonomous**.

6A.2. **Functionals**. The partial function $\alpha(\vec{x}, \vec{q})$ in (110) is an example of a **functional** in $\mathbb{N}$, i.e., of a partial function which accepts as input values natural numbers and partial functions of several variables, and (when it converges) yields a value in $\mathbb{N}$. Some simple, additional examples:

$$\begin{aligned} \alpha_1(\vec{x}) &= f(\vec{x}) && (\text{where } f : \mathbb{N}^n \rightharpoonup \mathbb{N}),\\ \alpha_2(p, r) &= p(0) && (p : \mathbb{N} \rightharpoonup \mathbb{N}, q : \mathbb{N}^2 \rightharpoonup \mathbb{N}),\\ \text{eval}^n(\vec{x}, p) &= p(\vec{x}) && (p : \mathbb{N}^n \rightharpoonup \mathbb{N}). \end{aligned}$$

The first of these makes clear that it is not necessary for a functional to have partial function arguments, i.e., *every partial function is a functional*; and α_2 is an example with no natural number arguments, only partial functions. Note also that for every partial function r, with $S(x) = x + 1$ and ε the "empty" partial function,

$$\alpha_2(S, r) = S(0) = 1, \quad \alpha_2(\varepsilon, r) = \varepsilon(0) = \uparrow .$$

The **call** or **evaluation** functionals $\text{eval}^n(\vec{x}, p)$ are perhaps the simplest non-trivial functionals. Some of their values are

$$\text{eval}^1(x, S) = S(x) = x + 1, \quad \text{eval}^3(x, y, z, P^3_2) = P^3_2(x, y, z) = y.$$

6A.3. Definition. A functional $\alpha(\vec{x}, \vec{q})$ is **recursive**, if it is computed by some recursive program E with external variables $\mathsf{q}_1, \dots, \mathsf{q}_l$, i.e., $\alpha = \alpha_E$ in (110), or, equivalently,

$$\alpha(\vec{x}, \vec{q}) = w \iff E, \vec{\mathsf{q}} := \vec{q} \vdash \mathsf{p}(\vec{x}) = w$$

for some internal variable p of E. We set

$$\mathcal{R} = \text{the class of recursive functionals on } \mathbb{N},$$

extending our earlier use of this notation.

6A.4. Lemma. (1) *The class $\mathcal{R}$ of recursive functionals includes all recursive partial functions and the call functionals* eval^n. *It is closed under substitution into its number arguments*

$$\alpha(\vec{x},\vec{p}) = \beta(\gamma_1(\vec{x},\vec{p}),\dots,\gamma_m(\vec{x},\vec{p}),\vec{p}),$$

branching

$$\alpha(\vec{x},\vec{p}) = \text{if } (\beta_1(\vec{x},\vec{p}) = 0) \text{ then } \beta_2(\vec{x},\vec{p}) \text{ else } \beta_3(\vec{x},\vec{p}),$$

and minimalization

$$\alpha(y,\vec{x},\vec{p}) = (\mu i \geq y)[\beta(i,\vec{x},\vec{p}) = 0].$$

(2) *The class $\mathcal{R}$ is closed under explicit definitions of the form*

$$\alpha(x_1,\dots,x_n,p_1,\dots,p_m) = \beta(x_{a_1},\dots,x_{a_k},p_{b_1},\dots,p_{b_l}), \tag{112}$$

where $a_1,\dots,a_k$, $b_1,\dots,b_l$ are sequences from the index sets $1,\dots,n$ and $1,\dots,m$ respectively.

Proof. The functional $\mathrm{eval}^n(\vec{x},q)$ is computed by the program

$$p(\vec{x}) = \mathsf{q}(\vec{x}).$$

The proofs of the other parts of (1) are exactly those of 2C.3 and 2D.1.

Part (2) justifies the addition of new variables and the *identification* of others, e.g., definitions of the form

$$\alpha(x,z,y,p,q,r) = \beta(y,x,y,y,p,r,r,p)$$

and its proof is easy, Problem x6A.1. ⊣

By this Lemma and the methods of Chapter 2, we can easily show the recursiveness of many simple functionals by simple computations. For example, if $\beta(y,x,q)$ is recursive, then

$$\alpha(x,y,p,q) = \text{if } (\beta(y,y,q) = 0) \text{ then } y+1 \text{ else } p(2y,x) \tag{113}$$

is also recursive by the following detailed proof: we first set

$$\begin{aligned}
\alpha_1(x,y,p,q) &= \beta(y,y,q) && \text{(explicit)}\\
\alpha_2(x,y,p,q) &= S(y) = y+1 && \text{(explicit)}\\
\beta_1(x,y,p) &= 2y && \text{(explicit)}\\
\beta_2(x,y,p) &= P^1_1(x) = x && \text{(explicit)}\\
\beta_3(x,y,p) &= \mathrm{eval}^2(\beta_1(x,y,p),\beta_2(x,y,p),p) = p(2y,x) && \text{(substitution)}\\
\alpha_3(x,y,p,q) &= \beta_3(x,y,p) = p(2y,x) && \text{(explicit),}
\end{aligned}$$

and then by branching,

$$\alpha(x,y,p,q) = \text{if } (\alpha_1(x,y,p,q) = 0) \text{ then } \alpha_2(x,y,p,q) \text{ else } \alpha_3(x,y,p,q).$$

In many cases, however, the easiest way to prove that some functional is recursive is to write some program which computes it.

Problems for Section 6A

x6A.1. Prove part (2) of Lemma 6A.4.

x6A.2. Write a program which computes the functional

$$\alpha(x, p) = \mu i\,[p(x + i) = 0].$$

6B. Non-deterministic recursion

In the definition 2B.4 of abstract machines we allowed non-deterministic transition relations, so that the arguments for the Church-Turing Thesis in section 3B are as broadly applicable as possible. We have not studied non-deterministic recursive programs up until now, but they have their uses.

6B.1. Definition. A **non-deterministic recursive program** is any system of definitions

$$(E)\qquad \begin{array}{ll} (e_0) & \mathsf{p}_0(\vec{x}_0) = E_0 \\ & \vdots \\ (e_k) & \mathsf{p}_k(\vec{x}_k) = E_k \end{array}$$

as in Definition 6A.1, where we now allow more than one definition for the internal function variables. We use the same notation for such, generalized programs, their *internal* and *external* variables are defined as before, and their *states*, *computations* and *terminal computations* are defined just as for the deterministic programs in section 2B, only we now allow many computations on the same input. It is characteristic of these programs that they do not necessarily determine a partial function $\overline{\mathsf{p}}$ for each internal variable p.

Every program can be viewed as non-deterministic since the definition does not require many definitions for some variable, it only allows them.

6B.2. **Some examples.** The program with two equations

$$(E_1)\qquad \mathsf{p}(x) = 0, \quad \mathsf{p}(x) = 1$$

does not compute a partial function: what would $\overline{\mathsf{p}}(0)$ be?

On the other hand, the program

$$(E_2)\qquad \begin{array}{ll} \mathsf{p}(x) = x, & \eta(y) = 0, \\ \mathsf{p}(x) = x + 1, & \theta(x) = \eta(\mathsf{p}(x)). \end{array}$$

computes the constant function $\overline{\theta}(x) = 0$, even though it assigns no partial function $\overline{\mathsf{p}}$ to the variable p. For each x, E_2 has two computations of the value $\overline{\theta}(x)$ with the computations illustrated in Figure 6.

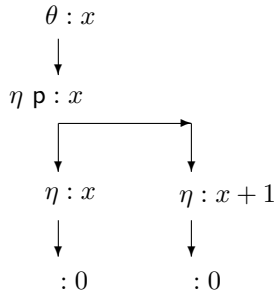
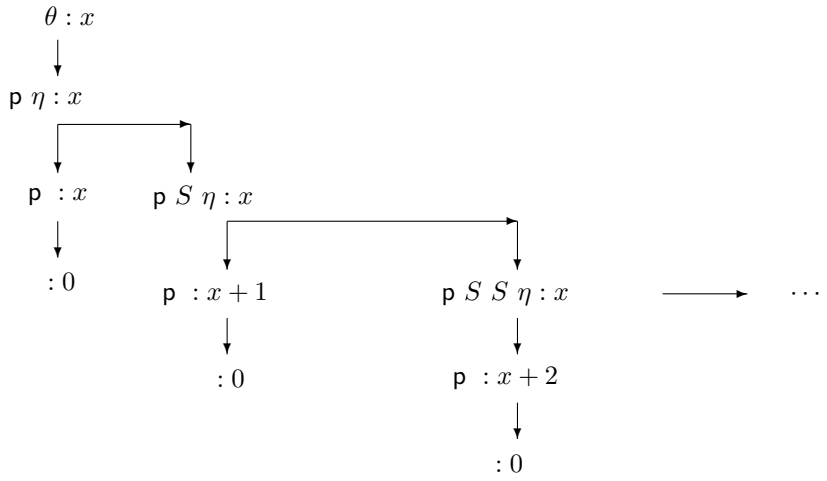


Figure 6. The computations of E_2.

Even more interesting is the program

$$(E_3) \qquad \begin{array}{ll} \eta(x) = x, & \mathsf{p}(x) = 0, \\ \eta(x) = S(\eta(x)), & \theta(x) = \mathsf{p}(\eta(x)) \end{array}$$

for which again $\overline{\theta}(x) = 0$, but now there are infinitely many computations on each input as follows:



Finally, consider the following example of a non-deterministic program with one external variable (q):

$$(E_4) \qquad \begin{array}{ll} \mathsf{p}_0(x) = \theta(\eta(x)), & \eta(x) = q(1), \\ \eta(x) = q(0), & \theta(x) = 0. \end{array}$$

If we assume that the computational semantics for non-deterministic programs with external function variables are basically the same as for deterministic ones, we would expect that E_4 computes the functional

$$\alpha_\vee(x,q) = \begin{cases} 0, & \text{if } q(0)\downarrow \vee\, q(1)\downarrow, \\ \uparrow, & \text{otherwise.} \end{cases} \tag{114}$$

After these preliminaries, we give the precise computational semantics for non-deterministic recursive programs:

6B.3. **Non-deterministically recursive functionals.** A partial function $f(\vec{x})$ is *non-deterministically recursive* if there exists a non-deterministic program E without external variables and some recursive variable p of E, such that for all $\vec{x}$,

$$\begin{aligned} f(\vec{x}) = w &\iff \mathbf{N}_0, E \vdash \mathsf{p}(\vec{x}) = w \\ &\iff \text{there exists a terminal computation of } E \\ &\qquad \text{with input } \mathsf{p} : \vec{x} \text{ and terminal state } : w. \end{aligned} \tag{115}$$

In other words, E computes $f(\vec{x})$ if, for every $\vec{x}$:

(1) There exists at least one terminal computation with input $\mathsf{p} : \vec{x}$ and terminal state $: f(\vec{x})$; and
(2) every terminal computation of E on input $\mathsf{p} : \vec{x}$ has terminal state $: f(\vec{x})$.

It is important to notice that the definition allows divergent computations (of "infinite length") as in E_3 above, which are simply disregarded.

The definition extends easily to functionals: in the simple case with just one variable over partial functions, a functional $\alpha(\vec{x}, q)$ is *non-deterministically recursive* if there is a non-deterministic program E with one external variable q and some internal variable p, so that for all $\vec{x}, p, w$,

$$\alpha(\vec{x}, q) = w \iff E, \mathsf{q} := q \vdash \mathsf{p}(\vec{x}) = w.$$

We set

$$\mathcal{R}_{nd} = \text{the set of non-deterministically recursive functionals.}$$

6B.4. Lemma. *A partial function $f(\vec{x})$ is recursive if and only if it is non-deterministically recursive.*

Proof. The coding of symbols, programs, states, computations, and generally of the whole theory of recursive programs in Section 3A is trivially extended to non-deterministic programs with just one difference in the details: in the definition of the relation $\mathrm{Prog}(e)$ we simply omit the last condition which forbids multiple definitions of the same function variable.

This allows the existence of many computations on the same input, i.e., it is possible that there exist y_1, y_2 such that

$$y_1 \neq y_2 \ \& \ T_n(e, \vec{x}, y_1) \ \& \ T_n(e, \vec{x}, y_2),$$

but the basic result does not change: if (115) holds for some f and some program E without external variables and with code e, then

$$f(\vec{x}) = w \iff (\exists y)[T_n(e, \vec{x}, y) \ \& \ U(y) = w],$$

so that the graph of $f(\vec{x})$ is Σ^0_1 and $f(\vec{x})$ is recursive. $\dashv$

For functionals, however, non-deterministic recursion is a proper extension of deterministic recursion, in fact the functional $\alpha_\vee(x, q)$ in (114) is not deterministically recursive. It is easy to see this directly, by considering the computations of any deterministic program which might compute $\alpha_\vee(x, q)$. But it is better to take a more general approach which helps clarify the notions.

6B.5. DEFINITION. A functional $\alpha(\vec{x}, p)$ is:

- **monotone** if for all partial functions p, q, and all $\vec{x}$, w,

$$\Big(\alpha(\vec{x}, p) = w \ \& \ p \sqsubseteq q\Big) \Longrightarrow \alpha(\vec{x}, q) = w;$$

- **continuous** if for every p and all $\vec{x}$, w,

$$\alpha(\vec{x}, p) = w \Longrightarrow (\exists r)\Big(r \sqsubseteq p \ \& \ \alpha(\vec{x}, r) = w \ \& \ r \text{ is finite}\Big),$$

  where a partial function is *finite* if it has a finite domain of convergence; and
- **deterministic** if for every p and all $\vec{x}$, w,

$$\alpha(\vec{x}, p) = w \Longrightarrow (\exists! r \sqsubseteq p)\Big(\alpha(\vec{x}, r) = w \ \& \ (\forall r' \sqsubseteq r)[\alpha(\vec{x}, r')\!\downarrow \implies r' = r]\Big).$$

The definitions are similar for functionals $\alpha(\vec{x}, \vec{p})$ with more variables, only messier to put down.

For example, the functional

$$\alpha(p) = \begin{cases} 0, & \text{if } p(0)\!\downarrow, \\ 1, & \text{otherwise,} \end{cases}$$

is not monotone; the functional

$$\beta(p) = \begin{cases} 0, & \text{if } p \text{ is total} \\ \uparrow, & \text{otherwise} \end{cases}$$

is not continuous; and the functional (114) is not deterministic, Problem x6B.2.

6B.6. Theorem. (1) *Every non-deterministically recursive functional is monotone and continuous.*

(2) *Every* (deterministically) *recursive functional is deterministic.*

It follows that there exist non-deterministically recursive functionals which are not recursive, e.g., $\alpha_\vee(x,q)$ *in* (114).

Proof. (1) For the monotonicity, we assume, that for some (possibly non-deterministic) program E with main symbol p_0 and an external variable q

$$\alpha(\vec{x},q)=w \iff E,\mathsf{q}:=q\vdash \mathsf{p}_0(\vec{x})=w,$$

that some computation

$$\mathsf{p}_0:\vec{x}\to\alpha_1:\beta_1\to\cdots\to\alpha_n:\beta_n\to:w \tag{116}$$

of E with the assignment $\mathsf{q}:=q$ yields the value $\alpha(\vec{x},q)=w$, and that $q\sqsubseteq r$; and we observe that the same sequence of states is a computation of E for the assignment $\mathsf{q}:=r$, because the transitions

$$\alpha^*\ \mathsf{q}:\vec{y}\ \beta^*\to\alpha^*:q(\vec{y})\ \beta^*$$

which call q are not stuck (otherwise the computation would stop), so $q(\vec{y})\!\downarrow$, and therefore $r(\vec{y})=q(\vec{y})$, so the same transition also holds for the computation with the assignment $\mathsf{q}:=r$.

In the same way, the computation (116) for the assignment $\mathsf{q}:=q$ is also a computation for the assignment $\mathsf{q}:=r$, where $r(\vec{y})$ converges only for the (finitely many) values of q which are called in (116), and so $\alpha(\vec{x},q)$ is continuous.

(2) If the program E is deterministic, then exactly one computation (116) computes the value $\mathsf{p}_0(\vec{x})=w$ for the assignment $\mathsf{q}:=q$, and the (finite) partial function r in the proof of continuity is the least $r\sqsubseteq q$ partial function such that $\alpha(\vec{x},r)\!\downarrow$, otherwise the computation (116) would terminate "earlier". ⊣

For the functionals in the class $\mathcal{R}_{nd}$ there exists a simple and useful normal form which uses the following coding.

6B.7. **Coding of finite functions and sets**. For every $z\in\mathbb{N}$, we set

$$d(z,i)=\begin{cases}(z)_i\dot{-}1, & \text{if } i<\mathrm{lh}(z)\ \&\ (z)_i>0,\\ \uparrow, & \text{otherwise,}\end{cases}$$
$$d_z(i)=d(z,i),$$
$$D_z=\{i\mid d_z(i)\!\downarrow\}.$$

The partial function $d(z,i)$ is recursive; the sequence

$$d_0,d_1,\dots$$

enumerates all finite, partial functions of one variable; and the sequence

$$D_0, D_1, \dots$$

enumerates all finite sets so that

$$i \in D_z \iff i < \mathrm{lh}(z) \ \&\ (z)_i > 0.$$

In particular, the relation $i \in D_z$ is primitive recursive.

6B.8. **Normal Form Theorem for $\mathcal{R}_{nd}$.** *A functional $\alpha(\vec{x}, p)$ is non-deterministically recursive, if and only if there exists a recursive relation $R(\vec{x}, w, z, y)$ such that*

$$\alpha(\vec{x}, p) = w \iff (\exists z)[d_z \sqsubseteq p \ \&\ (\exists y) R(\vec{x}, w, z, y)]. \tag{117}$$

PROOF. For the ($\Rightarrow$) direction with $\vec{x} = (x_1, \dots, x_n)$, set

$$\begin{aligned} T_n^*(e, \vec{x}, z, i, y) \iff\ & e \text{ is the code of some (possibly non-deterministic)} \\ & \text{recursive program } E \text{ and only } \mathsf{p}_i^1 \text{ is external in } E \\ & \text{and } y \text{ is the code of a terminal computation of } E \\ & \text{on input } \mathsf{p}_0^n : \vec{x} \text{ for the assignment } \mathsf{p}_i^1 := d_z. \end{aligned}$$

This relation is primitive recursive by the methods of 3A, and, easily, if E is such that

$$\alpha(\vec{x}, p) = w \iff E, \mathsf{p}_i^1 := p \vdash \mathsf{p}_0^n(\vec{x}) = w,$$

then

$$\alpha(\vec{x}, p) = w \iff (\exists z)[d_z \sqsubseteq p \ \&\ (\exists y)[T_n^*(e, \vec{x}, z, i, y) \ \&\ U(y) = w]].$$

For the other direction, we assume that the functional $\alpha(\vec{x}, p)$ satisfies the equivalence (117), we set

$$h(z, \vec{x}) = \Big(\mu y R(\vec{x}, (y)_0, z, (y)_1)\Big)_0,$$

and we observe that the functional

$$\beta(z, \vec{x}, p) = \text{if } (\forall i < \mathrm{lh}(z))[d_z(i)\downarrow \implies p(i) = d_z(i)] \text{ then } h(z, \vec{x}) \text{ else } \uparrow$$

is recursive, by the closure properties of the recursive functionals 6A.4. Let E be a (deterministic) program which computes the functional $\beta(z, \vec{x}, p)$ with main symbol p_0, so that

$$\beta(z, \vec{x}, \mathsf{p}) = w \iff E, \mathsf{p}_0 := \mathsf{p} \vdash \mathsf{p}_0(z, \vec{x}) = w.$$

Let E' be the non-deterministic program constructed by adding to E the following equations, with new variables:

$$\begin{aligned} \eta(t) &= 0 \\ \eta(t) &= S(\eta(t)) \\ \theta(\vec{x}) &= \mathsf{p}_0(\eta(0), \vec{x}). \end{aligned}$$

The terminal computations of E' starting with $\theta : \vec{x}$ are of the form

$$\begin{aligned} \theta &: \vec{x} \\ \mathsf{p}\ \eta &: 0\ \vec{x} \\ &\vdots \\ \mathsf{p} &: z\ \vec{x} \\ &\vdots \\ &: w \end{aligned}$$

and since, from the moment that the value z is computed on, the transitions which we added to E are not activated anymore, it follows that

$$w = \overline{\theta}(\vec{x}) = \beta(z, \vec{x}, p),$$

so that by the definitions,

$$d_z \sqsubseteq p \ \&\ (\exists y) R(\vec{x}, w, z, y), \tag{118}$$

i.e., $\alpha(\vec{x}, p) = w$. On the other hand, if $\alpha(\vec{x}, p) = w$, then there exists a z such that (118) holds, and with the transitions of $\eta(t)$ in E', there exists a computation of E' which reaches the point

$$\mathsf{p}_0 : z\ \vec{x}.$$

For example, if $z = 2$, the computation successively reaches the states

$$\begin{aligned} \theta &: \vec{x} \\ \mathsf{p}_0\ \eta &: 0\ \vec{x} \\ \mathsf{p}_0\ S\ \eta &: 0\ \vec{x} \\ \mathsf{p}_0\ S\ S\ \eta &: 0\ \vec{x} \\ \mathsf{p}_0\ S\ S &: 0\ \vec{x} \\ \mathsf{p}_0\ S &: 1\ \vec{x} \\ \mathsf{p}_0 &: 2\ \vec{x}. \end{aligned}$$

From this point on, the computation continues with the transitions of E and finally yields the correct value

$$\overline{\mathsf{p}}_0(z, \vec{x}) = \beta(z, \vec{x}, p) = \alpha(\vec{x}, p). \qquad \dashv$$

Problems for Section 6B

x6B.1. Determine which of the following two functionals are recursive or non-deterministically recursive and prove your answer:

$$\alpha(p) = \text{if } (\exists x)[p(x) = 1] \text{ then } 1 \text{ else } \uparrow$$
$$\beta(p) = \text{if } (\exists x)[p(x) = 1 \ \&\ (\forall i < x) p(i) = 0] \text{ then } 1 \text{ else } \uparrow$$

x6B.2. Prove by counterexamples that none of the conditions in definition 6B.5 follows from the other two.

x6B.3. Prove that for for any two sets A, B,

$A \leq_T B \iff$ there is some $\alpha(x,p) \in \mathcal{R}_{nd}$ such that $\chi_A(x) = \alpha(x, \chi_B)$.

HINT: Appeal to the Normal Form Theorem 6B.8.

6B.9. DEFINITION (**Many-valued non-deterministic recursion**). The non-deterministic program

$$\mathsf{p}(x) = 0, \quad \mathsf{p}(x) = 1,$$

computes no partial function, because, obviously, for every x it has two computations

$$\mathsf{p} : x \to^* \ : 0 \quad \mathsf{p} : x \to^* \ : 1,$$

and so it does not determine a specific value $\overline{\mathsf{p}}(x)$. It is, however, natural to consider as *the value* $\overline{p}(x)$ computed by E the set $\{0,1\}$ of the two values 0 and 1, and the next definition makes this notion rigorous.

For each function variable p in a given non-deterministic recursive program E (without external variables), we set

$$\begin{aligned}\widetilde{\mathsf{p}}(\vec{x}) = \{w \mid & w \in \mathbb{N} \ \& \text{ there exists a computation } \mathsf{p} : \vec{x} \to \cdots \to : w \text{ of } E \\ & \text{or } w = \uparrow \text{ and there exists a divergent computation of } E \\ & \mathsf{p} : \vec{x} \to \alpha_1 : \beta_1 \to \cdots\}.\end{aligned}$$

For example, the "value" of the program

$$\mathsf{p}(x) = 0, \quad \mathsf{p}(x) = 1, \quad \mathsf{p}(x) = \mathsf{p}(x)$$

is $\widetilde{\mathsf{p}}(x) = \{0, 1, \uparrow\}$.

x6B.4. Give examples of non-deterministic programs with main symbol p for which:

(1) $\widetilde{\mathsf{p}}(x) = \mathbb{N} \cup \{\uparrow\}$.
(2) $\widetilde{\mathsf{p}}(x) = \{0, 1, \ldots, x\}$.

x6B.5. Let E_+ be a program which computes addition with main function symbol $+$, and $letE$ the non-deterministic extension of E_+ by the definitions

$$\mathsf{p}(x) = 1, \quad \mathsf{p}(x) = \mathsf{p}(x) + \mathsf{p}(x).$$

What is the set $\widetilde{\mathsf{p}}(0)$;

x6B.6. Let E_2 be a program which computes the function $2x$ and let E be the non-deterministic extension of E_2 by the definitions

$$\mathsf{p}(x) = 1, \quad \mathsf{p}(x) = 2\mathsf{p}(x).$$

What is the set $\widetilde{\mathsf{p}}(0)$;

x6B.7*. Prove that if for some non-deterministic program E with main symbol p the set $\widetilde{\mathsf{p}}(x)$ is infinite, then $\uparrow \in \widetilde{\mathsf{p}}(x)$, i.e., there exists some divergent computation of E starting with the state $\mathsf{p} : x$. Hint: The proof requires an application of *König's Lemma*.

6C. The 1st Recursion Theorem

The result in this section complements the 2nd Recursion Theorem. It is especially important for the foundations of recursion theory.

6C.1. Lemma. *For each non-deterministically recursive functional $\alpha(\vec{x}, \vec{p})$, the partial function*

$$f(\vec{x}, e_1, \dots, e_l) = \alpha(\vec{x}, \varphi_{e_1}, \dots, \varphi_{e_l})$$

is recursive.

Proof. For the case with only one function variable, where the notation is simpler, we have from the Normal Form Theorem 6B.8 an equivalence

$$\alpha(\vec{x}, p) = w \iff \exists z)[d_z \sqsubseteq p \ \& \ (\exists y) R(\vec{x}, w, z, y)],$$

where $R(\vec{x}, w, z, y)$ is a recursive. So

$$\begin{aligned} f(\vec{x}, e) = w &\iff (\exists z)[d_z \sqsubseteq \varphi_e \ \& \ (\exists y) R(\vec{x}, w, z, y)] \\ &\iff (\exists z)[(\forall i < \mathrm{lh}(z)[d_z(i)\downarrow \implies \varphi_e(i) = d_z(i)] \\ &\qquad\qquad \& \ (\exists y) R(\vec{x}, w, z, y)], \end{aligned}$$

and $f(\vec{x}, e)$ is recursive because its graph is Σ^0_1. ⊣

6C.2. **The 1st Recursion Theorem.** *For every monotone and continuous functional $\alpha(x_1, \dots, x_n, p)$ where p is an n-place function variable, there exists a least solution $\overline{p}$ of the recursive equation*

$$p(\vec{x}) = \alpha(\vec{x}, p),$$

which is characterized by the conditions

(1) *for all $\vec{x}$, $\overline{p}(\vec{x}) = \alpha(\vec{x}, \overline{p})$, and*

(2) *for every $q : \mathbb{N}^n \rightharpoonup \mathbb{N}$,*

$$\text{if } (\forall \vec{x}, w)[\alpha(\vec{x}, q) = w \Longrightarrow q(\vec{x}) = w], \text{ then } \overline{p} \sqsubseteq \overline{q}.$$

Furthermore, if $\alpha(\vec{x}, p)$ is non-deterministically recursive, then $\overline{p}$ is recursive.

Proof. We first observe that the *uniqueness* of a partial function $\overline{p}$ which satisfies (1) and (2) of the theorem is trivial; because if $\overline{p}$ has these two properties and $\overline{p}'$ the corresponding

(1)′ $\overline{p}'(\vec{x}) = \alpha(\vec{x}, \overline{p}')$,
(2)′ for every $q : \mathbb{N} \rightharpoonup \mathbb{N}$,

$$(\forall \vec{x}, w)[\alpha(\vec{x}, q) = w \Longrightarrow q(\vec{x}) = w] \Longrightarrow \overline{p}' \sqsubseteq q,$$

then $\overline{p} \sqsubseteq \overline{p}'$ by (1)′ and (2) with $q = \overline{p}'$, and $\overline{p}' \sqsubseteq \overline{p}$ by (1) and (2)′ with $q = \overline{p}$. So it remains to show the existence of a partial function $\overline{p}$ which satisfies (1) and (2) and its recursiveness, in the case that $\alpha(\vec{x}, p)$ is non-deterministically recursive.

We set

$$\overline{p}^0(\vec{x}) = \uparrow \text{ (so } \overline{p} \text{ is the totally undefined partial function)},$$

and, recursively,

$$\overline{p}^{n+1}(\vec{x}) = \alpha(\vec{x}, \overline{p}^n).$$

Lemma. $\overline{p}^0 \sqsubseteq \overline{p}^1 \sqsubseteq \overline{p}^2 \sqsubseteq \cdots$,

so that for some partial function $\overline{p} : \mathbb{N} \rightharpoonup \mathbb{N}$,

$$\overline{p}(\vec{x}) = w \iff (\exists n)[\overline{p}^n(\vec{x}) = w]. \tag{119}$$

Proof. We show by induction on n that $\overline{p}^n \sqsubseteq \overline{p}^{n+1}$. The basis, $\overline{p}^0 \sqsubseteq \overline{p}^1$ is obvious, because $\overline{p}^0 \sqsubseteq q$, for every q. For the induction step:

$$\begin{aligned}
\overline{p}^{n+1}(\vec{x}) = w &\Longrightarrow \alpha(\vec{x}, \overline{p}^n) = w \qquad \text{by the definition} \\
&\Longrightarrow \alpha(\vec{x}, \overline{p}^{n+1}) = w \\
&\qquad\qquad \text{by the induction hypothesis } \overline{p}^n \sqsubseteq \overline{p}^{n+1} \\
&\qquad\qquad \text{and the monotonicity of } \alpha(\vec{x}, p) \\
&\Longrightarrow \overline{p}^{n+2}(\vec{x}) = w \qquad \text{by the definition.} \qquad \dashv \text{(Lemma)}
\end{aligned}$$

Proof of (1). We need to show that for all $\vec{x}$ and w,

$$\overline{p}(\vec{x}) = w \iff \alpha(\vec{x}, \overline{p}) = w,$$

and we will verify separately the two implications.

For $\overline{p}(\vec{x}) = w \Longrightarrow \alpha(\vec{x}, \overline{p}) = w$ first, we compute:

$$\begin{aligned}
\overline{p}(\vec{x}) = w &\Longrightarrow (\exists n)[\overline{p}^{n+1}(\vec{x}) = w] \quad \text{(by the definition)} \\
&\Longrightarrow (\exists n)[\alpha(\vec{x}, \overline{p}^n)) = w] \quad \text{(by the definition)} \\
&\Longrightarrow \alpha(\vec{x}, \overline{p}) = w \qquad \text{(monotonicity of } \alpha(p)).
\end{aligned}$$

For the converse direction, assume that $\alpha(\vec{x}, \overline{p}) = w$. By the continuity of $\alpha(\vec{x}, p)$, it follows that there exists some finite, partial, function $p^* \sqsubseteq \overline{p}$, with domain of convergence

$$\{\vec{x}_0, \ldots, \vec{x}_{k-1}\} = \{\vec{x} \mid p^*(\vec{x}) \downarrow\},$$

such that

$$\alpha(\vec{x}, p^*) = w. \tag{120}$$

By the definition of $\overline{p}$, for every $i < k$, there exists some n_i, such that

$$p^*(\vec{x}_i) = \overline{p}(\vec{x}_i) = \overline{p}^{n_i}(\vec{x}_i) \quad (i < k),$$

and if $n = \max\{n_0, \ldots, n_{k-1}\} + 1$, then

$$p^*(\vec{x}_i) = \overline{p}^{n}(\vec{x}_i) \quad (i < k),$$

i.e., $p^* \sqsubseteq \overline{p}^n$. The monotonicity of $\alpha(\vec{x}, p)$ now implies that

$$\alpha(\vec{x}, p^*) = w \Longrightarrow \alpha(\vec{x}, \overline{p}^n) = w \Longrightarrow \overline{p}^{n+1}(\vec{x}) = w$$

which with (120) completes the proof of (1).

Proof of (2). Suppose that for some $q : \mathbb{N} \rightharpoonup \mathbb{N}$

$$(\forall \vec{x}, w)[\alpha(\vec{x}, q) = w \Longrightarrow q(\vec{x}) = w].$$

The required $\overline{p} \sqsubseteq q$ follows from

$$\overline{p}^n \sqsubseteq q \quad (n \in \mathbb{N})$$

which obviously holds for $n = 0$. Inductively,

$$\begin{aligned}
\overline{p}^{n+1}(\vec{x}) = w &\Longrightarrow \alpha(\vec{x}, \overline{p}^n) = w && \text{by the definition} \\
&\Longrightarrow \alpha(\vec{x}, q) = w && \text{by the induction hypothesis} \\
& && \text{and the monotonicity of } \alpha \\
&\Longrightarrow q(\vec{x}) = w && \text{by the hypothesis for } q.
\end{aligned}$$

Finally, if $\alpha(\vec{x}, p)$ is non-deterministically recursive, then the partial function

$$f(e, \vec{x}) = \alpha(\vec{x}, \varphi_e)$$

is recursive by Lemma 6C.1, so that for some number $\widehat{f}$,

$$\{S_1^1(\widehat{f}, e)\}(\vec{x}) = \{\widehat{f}\}(e, \vec{x}) = \alpha(\vec{x}, \varphi_e).$$

It follows that if e_0 is some code of the empty partial function $\overline{p}^0(\vec{x}) = \uparrow$ and we set, recursively,

$$g(0) = e_0, \quad g(n+1) = S_1^1(\widehat{f}, g(n)),$$

then, for every n, $g(n)$ is a code of $\overline{p}^n$; so

$$\overline{p}(\vec{x}) = w \iff (\exists n)[\{g(n)\}(\vec{x}) = w],$$

and $\overline{p}$ is recursive, since its graph is Σ_1^0. $\dashv$

6D. Effective operations

In this and the next section we consider functionals $\alpha(\vec{x}, \vec{p})$ where the variables $p_1, \dots, p_m$ range over *recursive* (not arbitrary) partial functions, so that α can "call" its variables by their codes, i.e., "by name" in the terminology of programming languages. It is customary to call these functionals "operations".

6D.1. Definition. An **operation** (on the recursive partial functions) is any partial function

$$\alpha : \mathbb{N}^n \times \mathbb{P}_{k_1} \times \cdots \times \mathbb{P}^r_{k_m} \rightharpoonup \mathbb{N},$$

where $\mathbb{P}^r_k$ is the set of all recursive partial functions of k variables,

$$\mathbb{P}^r_k = \{\varphi^k_e \mid e \in \mathbb{N}\};$$

and the operation α is **effective** if the partial function

$$f(\vec{x}, e_1, \dots, e_m) = \alpha(\vec{x}, \varphi_{e_1}, \dots, \varphi_{e_m}) \tag{121}$$

is recursive.

We notice that the partial function f which computes the operation α satisfies the following **invariance property**:

$$\varphi_{e_1} = \varphi_{z_1}, \dots, \varphi_{e_m} = \varphi_{z_m} \Longrightarrow f(\vec{x}, e_1, \dots, e_m) = f(\vec{x}, z_1, \dots, z_m); \tag{122}$$

and (obviously) every recursive partial function f which satisfies (122) computes the operation

$$\alpha(\vec{x}, \varphi_{e_1}, \dots, \varphi_{e_m}) = f(\vec{x}, e_1, \dots, e_m).$$

The basic theorem of this section is that the effective operations (essentially) coincide with the non-deterministically recursive functionals, and one direction of this fact follows immediately from 6C.1. The key for the converse direction is the following

6D.2. Lemma. *Every effective operation is monotone and continuous.*

Proof. We consider only operations $\alpha : \mathbb{P}^r_1 \rightharpoonup \mathbb{N}$, the proof of the general case being only notationally more complex.

For the proof of monotonicity, let $p \sqsubseteq q$, where

$$p = \varphi_e \text{ and } q = \varphi_m,$$

and let $\widehat{f}$ a code of a partial function which computes α, i.e., for every z,

$$\alpha(\varphi_z) = \{\widehat{f}\}(z). \tag{123}$$

We also assume that

$$\alpha(\varphi_e) = w,$$

and we need to show $\alpha(\varphi_m) = w$.

The relation

$$R(z,x,v) \iff \varphi_e(x) = v \text{ or } \Big(\{\widehat{f}\}(z) = w \ \&\ \varphi_m(x) = v\Big)$$

is semirecursive; the hypothesis $\varphi_e \sqsubseteq \varphi_m$ implies that

$$R(z,x,v) \Longrightarrow \varphi_m(x) = v;$$

so $R(z,x,v)$ is the graph of some partial, recursive function $g(z,x)$; and then the 2nd Recursion Theorem implies that $\varphi_{z^*}(x) = g(z^*,x)$ for some number z^*, so that

$$\varphi_{z^*}(x) = v \iff \varphi_e(x) = v \text{ or } \Big(\{\widehat{f}\}(z^*) = w \ \&\ \varphi_m(x) = v\Big). \tag{124}$$

We now observe the following:

(1a) $\alpha(\varphi_{z^*}) = \{\widehat{f}\}(z^*) = w$. Because if this is not true, then $\varphi_{z^*} = \varphi_e$ by (124) and so $\alpha(\varphi_{z^*}) = \alpha(\varphi_e) = w$.

(1b) $\varphi_{z^*} = \varphi_m$, directly from the hypothesis $\varphi_e \sqsubseteq \varphi_m$ and (1a).

It follows that $\alpha(\varphi_m) = \alpha(\varphi_{z^*}) = w$.

The construction for the proof of continuity is a slight variation of the one we used for monotonicity. First we find from the 2nd Recursion Theorem some z^* such that

$$\begin{aligned}\varphi_{z^*}(x) = v \\ \iff (\forall u \le x)\neg[T_1(\widehat{f},z^*,u) \ \&\ U(u) = w] \ \&\ \varphi_e(x) = v,\end{aligned} \tag{125}$$

and we observe:

(2a) $\alpha(\varphi_{z^*}) = w$. Because in the opposite case,

$$(\forall u)\neg[T_1(\widehat{f},z^*,u) \ \&\ U(u) = w],$$

so for every x,

$$(\forall u \le x)\neg[T_1(\widehat{f},z^*,u) \ \&\ U(u) = w],$$

and so, by (125), $\varphi_{z^*} = \varphi_e$ and $\alpha(\varphi_{z^*}) = \alpha(\varphi_e) = w$.

(2b) $\varphi_{z^*} \sqsubseteq \varphi_e$, directly from (125).

(2c) The partial function φ_{z^*} is finite, as it converges only if

$$x < (\mu u)[T_1(\widehat{f},z^*,u) \ \&\ U(u) = w]. \tag{126}$$

⊣

6D.3. **The Myhill-Shepherdson Theorem.** *An operation α is effective if and only if it is the restriction to the recursive partial functions of some non-deterministically recursive functional, i.e., if for some $\beta \in \mathcal{R}_{nd}$,*

$$\alpha(\vec{x},\varphi_{e_1},\dots,\varphi_{e_m}) = \beta(\vec{x},\varphi_{e_1},\dots,\varphi_{e_m}).$$

Proof. One direction has been proved in Lemma 6C.1. For the other, we consider only operations $\alpha(p)$ with one, one-place variable. By the Normal Form Theorem 6B.8, it suffices to define a semirecursive relation $R(x, w)$, such that

$$\alpha(p) = w \iff (\exists x)[d_x \sqsubseteq p \ \&\ R(x, w)]. \tag{127}$$

If $\widehat{d}$ is a code of the partial function $d(z, i)$ which enumerates all finite, partial functions, then

$$d_z(i) = \{\widehat{d}\}(z, i) = \{S_1^1(\widehat{d}, z)\}(i),$$

and so, if the partial, recursive function $f(e)$ computes $\alpha(p)$, then the relation

$$R(z, w) \iff \alpha(d_z) = w \iff f(S_1^1(\widehat{d}, z)) = w$$

is semirecursive, and (127) follows with this $R(z, w)$ by Lemma 6D.2. ⊣

6D.4. Corolary (Rice-Shapiro). *A recursive partial function $f(e)$ satisfies*

$$W_e = W_m \Longrightarrow f(e) = f(m),$$

if and only if there is a semirecursive relation $R(x, w)$ such that

$$f(e) = w \iff (\exists x)[D_x \subseteq W_e \ \&\ R(x, w)],$$

where the enumeration $D_0, D_1, \ldots$ of the finite sets has been defined in 6B.7.

We leave the proof for Problem x6D.6.

Problems for Section 6D

x6D.1. Let $f(z)$ be a recursive partial function such that

$$[f(e)\downarrow \ \&\ W_e = W_m] \Longrightarrow f(m)\downarrow;$$

show that for every e,

$f(e)\downarrow \Longrightarrow$ there exists a finite W_z, such that $W_z \subseteq W_e$ and $f(z)\downarrow$.

x6D.2. Let $f(e)$ be a recursive partial function such that $f(e) \leq 5$ for every code e of a total, one-place function φ_e. Is it true or false that *there must be some m such that φ_m is not total, but $f(m)\downarrow$ and $f(m) \leq 5$?*. Prove your answer.

x6D.3. Let $f(e)$ be a recursive partial function such that

$$W_e = \emptyset \Longrightarrow f(e)\downarrow .$$

(1) Prove that for some $W_e \neq \emptyset$, $f(e)\downarrow$.

(2) Prove that for every m, there exists some e such that

$$W_e = W_m \text{ and } f(e)\downarrow .$$

x6D.4. Prove that for every effective operation $\alpha(x, p)$, the recursive equation

$$p(x) = \alpha(x, p)$$

has a recursive (partial) solution.

x6D.5. Prove that for every effective operation $\alpha(x, p)$, the recursive equation

$$p(x) = \alpha(x, p)$$

has a least solution, which is recursive.

x6D.6. Prove the Rice-Shapiro Theorem 6D.4.

x6D.7 (**Rice's theorem**). Prove that if a total function $f(e)$ satisfies the invariance property

$$W_e = W_m \Longrightarrow f(e) = f(m),$$

then $f(e)$ is constant.

x6D.8*. (1) Prove that there exists a recursive partial function $f(e)$ such that for every e,

$$f(e)\downarrow \iff (\exists x)[\varphi_e(x)\downarrow], \tag{128}$$

$$(\exists x)[\varphi_e(x)\downarrow] \implies \varphi_e(f(e))\downarrow . \tag{129}$$

(2) Prove that there is no partial recursive function $f(e)$ which satisfies the conditions (128), (129), and in addition

$$\varphi_e = \varphi_m \Longrightarrow f(e) = f(m). \tag{130}$$

6E. Kreisel-Lacombe-Shoenfield and Friedberg

The basic message of 6D.3 is that there is no way to use a program P (or a code of it) which computes a recursive partial function p in the computation of properties of p other than the obvious: in the process of the computation we can use P to compute any value $p(u)$ of p we need. The corresponding problem for effective operations on *total recursive functions* is more difficult and has a more interesting answer.

For each $k = 1, 2, \ldots$, let $\mathbb{F}^r_k$ be the set of all recursive (total) functions of k variables,

$$\mathbb{F}^r_k = \{\varphi^k_e \mid (\forall \vec{x})(\exists y) T_k(e, \vec{x}, y)\}.$$

In particular, $\mathbb{F}^r_1$ is the set of all *recursive sequences* of natural numbers.

6E.1. Definition. An **operation** (on the total recursive functions) is any partial function

$$\alpha : \mathbb{N}^n \times \mathbb{F}^r_{k_1} \times \cdots \times \mathbb{F}^r_{k_m} \rightharpoonup \mathbb{N};$$

and the operation α is *computable* if there exists some recursive partial function $f(\vec{x}, e_1, \ldots, e_m)$ such that

$$\varphi_{e_1}, \ldots, \varphi_{e_m} \in \mathbb{F}^r \Longrightarrow \alpha(\vec{x}, \varphi_{e_1}, \ldots, \varphi_{e_m}) = f(\vec{x}, e_1, \ldots, e_m), \tag{131}$$

where $\mathbb{F}^r = \bigcup_k \mathbb{F}^r_k$.

Here the partial function f which computes the operation α satisfies the *invariance property*:

$$\varphi_{e_1} = \varphi_{z_1}, \ldots, \varphi_{e_m} = \varphi_{z_m} \in \mathbb{F}^r \Longrightarrow f(\vec{x}, e_1, \ldots, e_m) = f(\vec{x}, z_1, \ldots, z_m) \tag{132}$$

which is significantly weaker than the corresponding property (122) for operations on the recursive partial functions. For example, the partial function

$$f(e) = \begin{cases} 1, & \text{if } (\forall i \leq e)[\varphi_e(i) = 0], \\ \uparrow, & \text{otherwise} \end{cases}$$

satisfies (131) and computes (rather unnaturally) the operation

$$\alpha(p) = 1$$

on $\mathbb{F}^r_1$). It does not satisfy (122) and does not compute any operation on the space $\mathbb{P}^r$.

There are two obvious questions, for the simpler case of operations $\alpha : \mathbb{F}^r_1 :\rightharpoonup \mathbb{N}$:

6E.2. **Question 1.** *Can we find, for each effective operation $\alpha : \mathbb{F}^r_1 \rightharpoonup \mathbb{N}$, some recursive, partial function which computes it according to* (131) *and which also satisfies the strong invariance property* (122)*?*

Equivalently, as we will see:

6E.3. **Question 2.** *Does there exist a recursive* (*or non-deterministically recursive*) *functional $\alpha^* : \mathbb{P}_1 \rightharpoonup \mathbb{N}$ such that*

$$X \in \mathbb{F}^r_1 \Longrightarrow \alpha(X) = \alpha^*(X)?$$

The basic content of the Kreisel-Lacombe-Shoenfield Theorem and of Friedberg's counterexample which we will show in this section is that the answer is *positive* in both of these questions *for effective total operations* $\alpha : \mathbb{F}^r_1 \to \mathbb{N}$, for which

$$X \in \mathbb{F}^r_1 \Longrightarrow \alpha(X)\downarrow,$$

but (in general) *negative* for operations which diverge for some values of their variables.

6E.4. Lemma. *Let $\alpha(X)$ be an effective operation on the space $\mathbb{F}_1^r$ and $f(e)$ a recursive partial function which computes it,*

$$\varphi_e \in \mathbb{F}_1^r \Longrightarrow \alpha(\varphi_e) = f(e),$$

and let $X = \varphi_e \in \mathbb{F}_1^r$ be a recursive sequence such that

$$\alpha(X) = f(e) = w \in \mathbb{N}.$$

It follows that for every $k \in \mathbb{N}$, there exists a sequence $X_k : \mathbb{N} \to \mathbb{N}$ such that:

(1) $t \leq k \Longrightarrow X_k(t) = X(t)$.
(2) $\alpha(X_k) = \alpha(X) = w$.
(3) *X_k is eventually zero, i.e., for some l and all $t > l$, $X_k(t) = 0$.*

Proof. By the 2nd Recursion Theorem 4D.1, for every k, there exists a code $z = z(k)$ of a recursive partial function, such that

$$\varphi_z(t) = \begin{cases} \varphi_e(t), & \text{if } t \leq k \text{ or } (\forall u \leq t)\neg[T_1(\widehat{f}, z, u) \ \&\ U(u) = w], \\ 0, & \text{otherwise,} \end{cases} \tag{133}$$

where $\widehat{f}$ is a code of f, i.e., $f(e) = \varphi_{\widehat{f}}(e)$. We set

$$X_k(t) = \varphi_z(t)$$

and we verify successively the required properties.

(1) $\varphi_z(t)$ is a total, recursive function and, for every $t \leq k$, $\varphi_z(t) = \varphi_e(t)$, directly from (133).

(2) $(\exists u)[T_1(\widehat{f}, z, u) \ \&\ U(u) = w]$, i.e., $\alpha(\varphi_z) = f(e) = w$. If not, then

$$(\forall u)\neg[T_1(\widehat{f}, z, u) \ \&\ U(u) = w];$$

from this it follows that, for every t, $(\forall u \leq t)\neg[T_1(\widehat{f}, z, u) \ \&\ U(u) = w]$; so $\varphi_z = \varphi_e$, by (133), and $f(z) = f(e) = w$, by the hypothesis that f is $\mathbb{F}_1^r$-invariant.

(3) For every $t > (\mu u)[T_1(\widehat{f}, z, u) \ \&\ U(u) = w]$, $\varphi_z(t) = 0$, directly from (133). ⊣

The Lemma asserts that the *eventually zero* (and hence recursive) sequences occur "densely" in every set

$$V_w = \{X \in \mathbb{F}_1^r \mid \alpha(X) = w\} \quad (w \in \mathbb{N}),$$

i.e., for every $X \in V_w$, there exist eventually zero sequences in V_w which "agree" with X on arbitrarily long, initial segments. The eventually zero sequences are coded simply, by the (basically) same coding which we used for finite, partial functions:

6E.5. **Coding of eventually zero functions**. Let

$$c(x,i) = \begin{cases} (x)_i, & \text{if } i < \mathrm{lh}(x), \\ 0, & \text{otherwise,} \end{cases}$$
$$c_x(i) = c(x,i).$$

It follows that the function $c(x,i)$ is recursive and the sequence

$$c_0, c_1, \dots$$

enumerates all eventually zero sequences, so that

$$c_x(i) \neq 0 \Longrightarrow i < \mathrm{lh}(x).$$

Also, there exists a primitive recursive function

$$\iota(x) = S_1^1(\widehat{c}, x) \quad (\text{where } \varphi_{\widehat{c}}(x,t) = c(x,t)), \tag{134}$$

such that

$$c_x(t) = c(x,t) = \varphi_{\iota(x)}(t).$$

6E.6. THEOREM (Continuity of effective operations on $\mathbb{F}_1^r$). *For each effective operation $\alpha : \mathbb{F}_1^r \rightharpoonup \mathbb{N}$ and every $X \in \mathbb{F}_1^r$, if $\alpha(X)\downarrow$, then there exists some $k \in \mathbb{N}$, such that for every $Y \in \mathbb{F}_1^r$,*

$$\Big(\alpha(Y)\downarrow \ \ \& \ (\forall t \leq k)[Y(t) = X(t)]\Big) \Longrightarrow \alpha(Y) = \alpha(X).$$

In particular, if α is total, so that, for every $Y \in \mathbb{F}_1^r$, $\alpha(Y)\downarrow$, the conclusion takes the simpler form

$$(\forall t \leq k)[Y(t) = X(t)] \Longrightarrow \alpha(Y) = \alpha(X).$$

PROOF. Let $f(e)$ be a recursive, partial function which computes $\alpha(X)$, so that

$$\varphi_e = \varphi_m \in \mathbb{F}_1^r \Longrightarrow f(e) = f(m).$$

The idea of the construction is to find some z such that

$$\varphi_z(t) = \begin{cases} X(t), & \text{if in } \leq t \text{ "steps" } f(z) \text{ does not converge,} \\ c_x(t), & \text{otherwise} \end{cases}$$

where the eventually zero c_x is chosen (compatibly with the first case, if it exists) so that

$$\alpha(c_x)\downarrow \ \ \& \ \alpha(c_x) \neq \alpha(X).$$

If we achieve this, then we will have $f(z) = \alpha(X)$, with an argument which is by now familiar, and from this we will conclude that if

$$k = \text{the "number of steps" in which } f(z) \text{ converges,}$$

then *there is no* x such that

$$\alpha(c_x)\downarrow \ \ \& \ \alpha(c_x) \neq \alpha(X) \ \& \ (\forall t \leq k)[c_x(t) = X(t)];$$

from which, with Lemma 6E.4, we will conclude further that *there is no recursive sequence Y such that*

$$(\forall t \leq k)[Y(t) = X(t)] \ \& \ \alpha(Y)\downarrow \ \& \ \alpha(Y) \neq \alpha(X),$$

which is what we needed to prove. In detail:

By the 2nd Recursion Theorem, for every e, there is a code z, such that

$$\varphi_z(t) = \begin{cases} \varphi_e(t), & \text{if } f(e)\downarrow \ \& \ (\forall u \leq t)\neg[T_1(\widehat{f}, z, u) \ \& \ U(u) = f(e)], \\ c_{g(z,e)}(t), & \text{if } f(e)\downarrow \ \& \ (\exists u \leq t)[T_1(\widehat{f}, z, u) \ \& \ U(u) = f(e)], \\ \uparrow, & \text{otherwise, i.e., if } f(e)\uparrow, \end{cases} \tag{135}$$

where

$$\mathrm{comp}(\widehat{f}, z) = \mu y T_1(\widehat{f}, z, y)$$

is the length of the computation of $f(z)$ (if $f(z)\downarrow$) and

$$g(z,e) = (\nu x)[f(z)\downarrow \ \& \ f(\iota(x))\downarrow \ \& \ f(z) \neq f(\iota(x)) \ \& \ (\forall t \leq \mathrm{comp}(\widehat{f}, z))[c_x(t) = \varphi_e(t)]]. \tag{136}$$

Here $\iota(x)$ is from (134), so that $\varphi_{\iota(x)} = c_x$ and by the Σ^0_1-Selection Lemma 4A.7, $g(z,e)$ converges exactly if there exists some x such that

$$f(z)\downarrow \ \& \ f(\iota(x))\downarrow \ \& \ f(z) \neq f(\iota(x)) \ \& \ (\forall t \leq \mathrm{comp}(\widehat{f}, z))[c_x(t) = \varphi_e(t)],$$

and, when it converges, it chooses some $x = g(z,e)$ with these properties.

We now suppose that $X = \varphi_e \in \mathbb{F}^r_1$ and $\alpha(X) = f(e)\downarrow$.

(1) *$f(z)\downarrow$ and $f(z) = f(e)$*; otherwise $\varphi_z = \varphi_e \in \mathbb{F}^r_1$ and $f(z) = f(e)$, which is absurd. We set

$$k = \mathrm{comp}(\widehat{f}, z). \tag{137}$$

(2) *There is no eventually zero function c_x, such that*

$$\alpha(c_x)\downarrow \ \& \ f(\iota(x)) = \alpha(c_x) \neq f(e) \ \& \ (\forall t \leq \mathrm{comp}(\widehat{f}, z))[c_x(t) = \varphi_e(t)].$$

Because if such a function existed, then $g(z,e)\downarrow$ and $c_{g(z,e)}$ has this property, so that, by the construction, $\varphi_z = c_{g(z,e)}$ and

$$f(z) = f(\iota(g(z,e))) = \alpha(c_{g(z,e)}) \neq f(e),$$

which contradicts (1).

(3) *For every $Y \in \mathbb{F}^r_1$,*

$$[\alpha(Y)\downarrow \ \& \ (\forall t \leq k)[Y(t) = X(t)]] \Longrightarrow \alpha(Y) = \alpha(X).$$

This follows now from Lemma 6E.4: because if $\alpha(Y) = v \neq f(e)$, then there exists some eventually zero c_x which agrees with Y (and, therefore, and with X) for $t \leq k$, and for which $\alpha(c_x) = v \neq f(e)$, and that contradicts (2). $\dashv$

This theorem is the basic discovery and it is called "the Kreisel-Lacombe-Shoenfield Theorem", or it is attributed to the Russian mathematician Čeitin who proved it independently. The name, however, is more appropriate for the next stronger result:

6E.7. THEOREM (Kreisel-Lacombe-Shoenfield, Čeitin). *For every effective operation $\alpha : \mathbb{F}_1^r \rightharpoonup \mathbb{N}$ on the total recursive functions, there is a recursive functional $\alpha^* : \mathbb{P}_1 \rightharpoonup \mathbb{N}$, such that*

$$\Big(X \in \mathbb{F}_1^r \ \&\ \alpha(X)\!\downarrow\ \Big) \Longrightarrow \alpha(X) = \alpha^*(X). \tag{138}$$

In particular, if $\alpha : \mathbb{F}_1^r \to \mathbb{N}$ is total, then

$$X \in \mathbb{F}_1^r \Longrightarrow \alpha(X) = \alpha^*(X),$$

and the questions 6E.2 and 6E.3 have positive answers for total effective operations.

PROOF. Let $\widehat{f}$ be a code of some recursive, partial function which computes α, i.e.,

$$\varphi_e \in \mathbb{F}_1^r \Longrightarrow \alpha(\varphi_e) = f(e) = \varphi_{\widehat{f}}(e).$$

The crucial observation is that the proof of the Continuity Theorem 6E.6 is *constructive*, and specifically that the number k in (137) is the value of a recursive, partial function $\sigma(e)$, which converges when e is a code of a recursive sequence φ_e such that $\alpha(\varphi_e) = f(e)\!\downarrow$. The partial function $g(z, e)$ is recursive indeed, as a function of two variables, with the definition (136); by the version (97) of the 2nd Recursion Theorem with parameter, there exists some primitive recursive $h(e)$ such that (135) holds if we set

$$z = h(e);$$

and, finally, the number k that we need is computed by

$$k = \sigma(e) = \mathrm{comp}(\widehat{f}, h(e)). \tag{139}$$

We now reexamine the proof of 6E.6, to see what it gives us *without the hypothesis that φ_e is total*:

LEMMA. *For every e, with $z = h(e)$, if*

$$f(e)\!\downarrow\ \ \&\ f(e) = f(z)\ \&\ (\forall t \leq \sigma(e))\varphi_e(t)\!\downarrow,$$

then, for every X,

$$\Big(X \in \mathbb{F}_1^r\ \&\ \alpha(X)\!\downarrow\ \ \&\ (\forall t \leq \sigma(e))X(t) = \varphi_e(t)\Big) \Longrightarrow \alpha(X) = f(e).$$

Proof. Towards a contradiction, we assume the hypothesis for e and that for some $X \in \mathbb{F}_1^r$,

$$\alpha(X)\!\downarrow\ \ \&\ (\forall t \leq \sigma(e))[X(t) = \varphi_e(t)\ \&\ \alpha(X) \neq f(e)].$$

By lemma 6E.4 now, there exists some eventually zero c_x such that

$$\alpha(c_x) = \alpha(X) \neq f(e) \ \& \ (\forall t \leq \sigma(e))c_x(t) = \varphi_e(t);$$

and by the definition of $g(e,z)$ and (135), $c_{g(e,z)}$ has this property and $\varphi_z = c_{g(e,z)} \in \mathbb{F}_1^r$; so $f(z) = f(g(e,z)) \neq f(e)$, which contradicts the hypothesis.

The set

$$A = \{e \mid f(e)\downarrow \ \& \ f(e) = f(h(e)) \ \& \ (\forall t \leq \sigma(e))\varphi_e(t)\downarrow\},$$

of all numbers e which satisfy the hypothesis of the Lemma is semirecursive, so there is a (primitive) recursive relation $R(e,u)$ such that

$$e \in A \iff (\exists u)R(e,u).$$

The functional

$$\gamma(p) = (\mu y)[(\forall t \leq y)p(t)\downarrow \ \& \ R((y)_0,(y)_1)]$$

is (easily) recursive. It follows that

$$\beta(p) = f((\gamma(p))_0),$$

is also recursive, and it is not difficult, by the Lemma, to show now that

$$[X \in \mathbb{F}_1^r \ \& \ \alpha(X)\downarrow] \Longrightarrow \alpha(X) = \beta(X),$$

which is what we needed to show. ⊣

The rather careful formulation of the Theorem is necessary, because of the following counterexample:

6E.8. Theorem (Friedberg). *There exists an effective* (partial) *operation $\alpha : \mathbb{F}_1^r \rightharpoonup \mathbb{N}$ such that:*

(1) *$\alpha(X_0) = 1$, where, $X_0(t) = 0$, for every t.*

(2) *For every k, there exists some $X_k \in \mathbb{F}_1^r$ such that*

$$(\forall t \leq k)X_k(t) = 0 \text{ and } \alpha(X_k)\uparrow .$$

It follows that the operation α is not the restriction to $\mathbb{F}_1^r$ of any non-deterministically recursive functional.

Proof. The second claim follows from (1) and (2), because, if α were the restriction to $\mathbb{F}_1^r$ of some (non-deterministically) recursive β, then $\beta(X_0) = 1$; so the computation of the recursive machine which computes the value $\beta(X_0)$ terminates and, until it terminates, it calls finitely many values of X_0; and if k is the maximum number for which the computation used the value $X_0(k)$, then, obviously, the computation will terminate and will give the value 1 for every X such that $(\forall t \leq k)X(t) = 0$.

To construct α and prove the first proposition, we set:

$$e \in A \iff (\forall t \leq e)[\varphi_e(t) = 0], \tag{140}$$

$$e \in B \iff (\exists m \in A)(\exists k)(\forall t \leq k)\Big(\varphi_e(t) = \varphi_m(t) = 0 \ \& \ \varphi_e(k+1) = \varphi_m(k+1)\downarrow \ \& \ \varphi_e(k+1) \neq 0\Big), \tag{141}$$

$$f(e) = \text{if } (e \in A \cup B) \text{ then } 1 \text{ else } \uparrow . \tag{142}$$

Lemma. The partial function f is recursive and $\mathbb{F}_1^\tau$-invariant, i.e.,

$$\varphi_e = \varphi_m \in \mathbb{F}_1^\tau \Longrightarrow f(e) = f(m).$$

Proof. Suppose $\varphi_e = \varphi_m \in \mathbb{F}_1^\tau$, and $f(e) = 1$. We need to show that $f(m) = 1$.

Case 1. $\varphi_e = \varphi_m = X_0$. In this case $m \in A$, so $f(m) = 1$.

Case 2. $\varphi_e = \varphi_m \neq X_0$ and $e \in A$. Now (141) holds with $m = e$, $k = e$ and m in place of e, so $m \in B$ and $f(m) = 1$.

CASE 3. $\varphi_e = \varphi_m \neq X_0$ and $e \in B$. In this case there must exist "witnesses" $m \in A$ and k which satisfy (141) and verify that $e \in B$; the same witnesses satisfy (141) with m in place of e, so that $m \in B$ and $f(m) = 1$. $\dashv$ (Lemma)

The effective operation $\alpha : \mathbb{F}_1^\tau \rightharpoonup \mathbb{N}$ computed by f obviously has property (1) in the Theorem. To show (2), for any k, we set

$$C = \{e \in A \mid e \leq k \ \& \ (\forall t \leq k)\varphi_e(t) = 0 \ \& \ \varphi_e(k+1)\downarrow \ \& \ \varphi_e(k+1) \neq 0\}.$$

The set C is finite, and (if k is sufficiently large) non-empty, say

$$C = \{e_1, \ldots, e_n\};$$

We set

$$X_k(t) = \begin{cases} 0, & \text{if } t \leq k \text{ or } t > k+1, \\ \max\{\varphi_{e_1}(k+1), \ldots, \varphi_{e_n}(k+1)\} + 1, & \text{if } t = k+1, \end{cases}$$

so that, certainly, $(\forall t \leq k)[X_k(t) = 0]$, and it is enough to get a contradiction from the hypothesis that there exists some e such that

$$\varphi_e = X_k \ \& \ e \in A \cup B.$$

CASE 1. $\varphi_e = X_k$ and $e \in A$. By the definition of A,

$$\begin{aligned} e \in A &\Longrightarrow (\forall t \leq e)[\varphi_e(t) = 0] \\ &\Longrightarrow e \leq k \qquad \text{since } \varphi_e(k+1) = X_k(k+1) \neq 0 \\ &\Longrightarrow e \in C \end{aligned}$$

and the last is absurd, because $X_k(k+1) \neq \varphi_e(k+1)$ for every $e \in C$.

CASE 2. $\varphi_e = X_k$ and $e \in B$. There exist now witnesses $m \in A$ and k' verifying that $e \in B$, and $k' = k+1$, since $(\forall t \leq k)[X_k(t) = 0]$ and $X_k(k+1) \neq 0$. Also, $m \in C$, since $m \in A$ and $\varphi_m(k+1) \neq 0$. But these

lead to a contradiction as in the first case, since $X_k(k+1) \neq \varphi_m(k+1)$, for every $m \in A$. $\dashv$

Problems for Section 6E

x6E.1. Let $f(e)$ be a recursive partial function such that

$$(\forall x)[\varphi_e(x) = 0] \Longrightarrow f(e) = 3;$$

show that for every k, there exists some m, such that

(1) $(\forall x)[\varphi_m(x)\downarrow]$.
(2) $(\forall x \leq k)[\varphi_m(x) = 0]$.
(3) $(\exists x)[\varphi_m(x) \neq 0]$.
(4) $f(m) = 3$.

x6E.2. Is the following proposition true or not: Let $f(e)$ be a recursive partial function such that

$$(\forall x)[\varphi_e(x) = 0] \Longrightarrow f(e)\downarrow;$$

for every k, there exists some m, such that

(1) $(\forall x)[\varphi_m(x)\downarrow]$.
(2) $(\forall x \leq k)[\varphi_m(x) = 0]$.
(3) $(\exists x)[\varphi_m(x) \neq 0]$.
(4) $f(m)\downarrow$.

Prove your answer.