Part A

General Theory

Chapter 1
General Theory: Combinatorial Part

This chapter contains the basic definitions, examples, and constructions of the general theory of computations—and two theorems: In Section 1.6 we prove the simple representation theorem (1.6.3) which is used in Section 1.7 to prove a general version of the first recursion theorem (1.7.8, 1.7.9).

1.1 Basic Definitions

In a typical computation situation there are three separate parts. There is a *computing device* or *machine* M which acts upon an *input* σ from some fixed input alphabet or domain A_1. After "computing" for some time M may give an *output* z belonging to some fixed output alphabet or domain A_2.

The input σ is usually a finite sequence $\sigma = (x_1, \ldots, x_n)$ of elements from A_1, the output a single element from A_2. In many cases there may be practical advantages in having distinct input and output alphabets. In a theoretical analysis this is superfluous, hence we shall from now on assume that $A_1 = A_2$ is a fixed set of objects, the *computation domain.*

The computing device M is usually one of a class of similar computing "machines". It was a basic insight of the early theories of computation that the machines could be *coded* by elements of the computation domain. This idea will be our starting point: The basic axioms shall provide an analysis of the relation

$$\{a\}(\sigma) \simeq z,$$

which is *intended* to assert that the computing device named or coded by a and acting on the input sequence $\sigma = (x_1, \ldots, x_n)$ gives z as output.

1.1.1 Definition. A *computation domain* is a structure

$$\mathfrak{A} = \langle A, C; 0, 1\rangle$$

where A is a non-empty set, C is a subset of A, and 0, 1 are two designated elements of C.

C is called the set of *codes*. It may or may not be equal to A. In ordinary Turing machine theory on ω the natural numbers, the set of codes C is equal to ω, i.e. every element of the computation domain codes a Turing machine. In higher types the set of codes may still be ω, which is now a proper subset of the computation domain, which consists of numbers, functions, and functionals of various types.

In the general case we have given the two sets A and C such that $\varnothing \neq C \subseteq A$. Usually C contains (an isomorphic copy of) ω. At the present level of generality we shall not make this a part of the definition of computation domain. We also remark that in many examples the output set is (a subset of) the code set.

To facilitate the presentation we introduce some notational conventions. We use

$x, y, z, \ldots$	for elements in A.
$a, b, c, \ldots$	for elements in C.
$\sigma, \tau, \ldots$	for finite sequences from A.

In particular, we let σ, τ or (σ, τ) denote the concatenation of sequences. By (a, σ, z), where $a \in C$, $z \in A$ and $\sigma = (x_1, \ldots, x_n)$ is a finite sequence from A, we understand the sequence

$$(a, x_1, \ldots, x_n, z).$$

As usual $\mathrm{lh}(\sigma) =$ the length of the sequence σ.

1.1.2 Definition. Θ is a *computation set* over $\mathfrak{A}$ if Θ is a set of tuples (a, σ, z), where $a \in C$, $\sigma = (x_1, \ldots, x_n)$, each $x_i \in A$, $z \in A$, and $\mathrm{lh}(a, \sigma, z) \geqslant 2$.

Requiring $\mathrm{lh}(a, \sigma, z) \geqslant 2$ means that we have a code a and an output z present, but not necessarily an input sequence.

1.1.3 Remark. At this stage we need not make any requirement of single-valuedness, hence given a and σ there may be more than one z such that $(a, \sigma, z) \in \Theta$. However, in most cases we will require that Θ is single-valued, e.g. in analyzing the subcomputations of computations in functionals.

Let Θ be a computation set over the domain $\mathfrak{A}$. To every $a \in C$ and every natural number $n \geqslant 0$ we can associate a *partial multiple-valued* function (pmv function) $\{a\}_\Theta^n$ as follows

$$\{a\}_\Theta^n(\sigma) \simeq z \quad \text{iff} \quad \mathrm{lh}(\sigma) = n \quad \text{and} \quad (a, \sigma, z) \in \Theta.$$

A pmv function f is really a map from A, or a suitable cartesian product over A, to subsets of A (including the empty subset). The functional notation then has the following meaning

$$\begin{array}{rll} f(\sigma) \simeq z & \text{means} & z \in f(\sigma) \\ f(\sigma) = g(\sigma) & \text{means} & \forall z[f(\sigma) \simeq z \text{ iff } g(\sigma) \simeq z] \\ f(\sigma) = z & \text{means} & f(\sigma) = \{z\} \\ f \subseteq g & \text{means} & \forall\sigma\forall z[f(\sigma) \simeq z \Rightarrow g(\sigma) \simeq z]. \end{array}$$

By a *mapping* we understand a total, single-valued function.

1.1.4 Definition. Let Θ be a computation set over $\mathfrak{A}$. A pmv function f is Θ-*computable* if for some $\hat{f} \in C$ we have

$$f(\sigma) \simeq z \quad \text{iff} \quad (\hat{f}, \sigma, z) \in \Theta.$$

We call $\hat{f}$ a Θ-*code* for f and write $f = \{\hat{f}\}_\Theta^n$, where $n = \mathrm{lh}(\sigma)$ is the number of arguments of f.

A partial multiple-valued (pmv) functional on A

$$\varphi(\mathbf{f}, \sigma) = \varphi(f_1, \ldots, f_l, x_1, \ldots, x_m),$$

maps pmv functions on A and elements of A to subsets of A. φ is called *consistent* if

$$f_1 \subseteq g_1, \ldots, f_l \subseteq g_l, \varphi(\mathbf{f}, \sigma) \simeq z \quad \Rightarrow \quad \varphi(\mathbf{g}, \sigma) \simeq z.$$

1.1.5 Definition. Let Θ be a computation set over $\mathfrak{A}$. A pmv consistent functional φ is called *weakly* Θ-*computable* if there exists a code $\hat{\varphi} \in C$ such that for all $e_1, \ldots, e_l \in C$ and all sequences $\sigma = (x_1, \ldots, x_n)$ from A we have

$$\varphi(\{e_1\}_\Theta^{n_1}, \ldots, \{e_l\}_\Theta^{n_l}, \sigma) \simeq z \quad \text{iff} \quad \{\hat{\varphi}\}_\Theta^{l+n}(e_1, \ldots, e_l, \sigma) \simeq z.$$

We see that φ is weakly Θ-computable if we can calculate φ on Θ-computable functions by calculating on the codes of the functions.

The notion of *weak* Θ-computability presupposes a notion of *strong* Θ-computability. Such a notion will be introduced in Section 1.7 below. The distinction between weak and strong computability will also be of importance in analyzing computations in higher types.

We will now consider some specific functions and functionals.

1.1.6 Definition by Cases (on the code set C).

$$DC(x, a, b, c) = \begin{cases} 1, & \text{if not all } a, b, c \in C \\ a, & \text{if } x = c \text{ and all } a, b, c \in C \\ b, & \text{if } x \neq c \text{ and all } a, b, c \in C. \end{cases}$$

Outright definition by cases makes equality on A Θ-computable. This we may not always want, e.g. in higher type theories. We note that DC is a mapping, i.e. total and single-valued.

1.1.7 Composition.

$$\mathbf{C}^n(f, g, \sigma) = f(g(\sigma), \sigma),$$

where $n = \mathrm{lh}(\sigma)$.

1.1.8 Permutation.

$$\mathbf{P}^m_{n,j}(f, \sigma, \tau) = f(\sigma^j).$$

Here $n, m \geqslant 0$, $\mathrm{lh}(\sigma) = n$, $\mathrm{lh}(\tau) = m$, $0 \leqslant j < n$, and $(x_1, \ldots, x_n)^j = (x_{j+1}, x_1, \ldots, x_j, x_{j+2}, \ldots, x_n)$. This functional performs many tasks, e.g. $\mathbf{P}^0_{n,0}$ is the evaluation functional $\mathbf{P}^0_{n,0}(f, \sigma) \simeq f(\sigma)$, as $\sigma^0 = \sigma$.

Next we consider a property which a computation set Θ on $\mathfrak{A}$ may or may not have.

1.1.9 Iteration Property. For each $m, n \geqslant 0$ there exists a mapping S^n_m such that for all $a \in C$ and sequences σ from C and all τ from A,

$$\{a\}^{n+m}_\Theta(\sigma, \tau) = \{S^n_m(a, \sigma)\}^m_\Theta(\tau),$$

where $n = \mathrm{lh}(\sigma)$ and $m = \mathrm{lh}(\tau)$.

1.1.10 Definition. Let Θ be a computation set on the domain $\mathfrak{A}$. Θ is called a *precomputation theory* on $\mathfrak{A}$ if

(i) for each n, j $(0 \leqslant j < n)$ and m, DC, $\mathbf{C}^n$, and $\mathbf{P}^m_{n,j}$ are Θ-computable with Θ-codes d, c_n, and $p_{n,j,m}$, respectively;

(ii) Θ satisfies the iteration property, i.e. for each n, m there is a Θ-code $s_{n,m}$ for a mapping S^n_m with property 1.1.9 above.

We make some remarks: **(1)** $d, c_n, p_{n,j,m}$ are all different members of C, hence C is infinite. **(2)** The use of functionals could be eliminated, we might directly refer to the codes. **(3)** If $\langle N, s\rangle$ (i.e. the natural numbers (or a copy) with the successor function is in the structure $\mathfrak{A}$, $N \subseteq C \subseteq A$, we may require that the codes $c_n, p_{n,j,m}$, and $s_{n,m}$ are Θ-computable mappings of the parameters n, j, m. This is a uniformity requirement which will be important in later sections. **(4)** $P^0_{n,0}$ is the evaluation functional. The Θ-computability of it gives us the following enumeration property: There is for each natural number n a code $p_{n,0,0} \in C$ such that for all $a \in C$ and all σ from A

$$\{p_{n,0,0}\}^{n+1}_\Theta(a, \sigma) \simeq \{a\}^n_\Theta(\sigma).$$

1.2 Some Computable Functions

We shall give several examples of functions which are Θ-computable for any precomputation theory Θ.

1.2.1 The Characteristic Function of the Code Set C. This function is defined as follows:

$$\chi_C^{(n)}(y, \sigma) = \begin{cases} 0, & \text{if } y \in C \\ 1, & \text{if } y \notin C \end{cases}$$

where $n = \mathrm{lh}(\sigma)$. A code k_n for $\chi_C^{(n)}$ is given by

$$k_n = S_{n+1}^1(p_{1,0,n}, S_1^3(d, 0, 0, 0)),$$

as the following calculation shows:

$$\begin{aligned} \{k_n\}_\Theta^{n+1}(y, \sigma) &= \{S_{n+1}^1(p_{1,0,n}, S_1^3(d, 0, 0, 0))\}_\Theta^{n+1}(y, \sigma) \\ &= \{p_{1,0,n}\}_\Theta^{n+1}(S_1^3(d, 0, 0, 0), y, \sigma) \\ &= \{S_1^3(d, 0, 0, 0)\}_\Theta^1(y) \\ &= DC(0, 0, 0, y) = \begin{cases} 0, & \text{if } y \in C \\ 1, & \text{if } y \notin C. \end{cases} \end{aligned}$$

Most such calculations will be omitted in the sequel.

1.2.2 Equality on C. For each $a \in C$ we can define a function E_a such that

$$E_a(x) = \begin{cases} 0, & \text{if } x = a \\ 1, & \text{if } x \neq a. \end{cases}$$

We see that $DC(x, 0, 1, a)$ almost does what we want. It has to be "straightened out" by using $\mathbf{P}_{n,j}^m$ a number of times in conjunction with the iteration property.

1.2.3 Identity Function on C. For each n, let

$$I_n(y, \sigma) = \begin{cases} y, & \text{if } y \in C \\ 1, & \text{if } y \notin C, \end{cases}$$

where $n = \mathrm{lh}(\sigma)$. The following is code for I_n

$$i_n = S_{n+1}^3(p_{n+3,2,0}, S_{n+3}^2(c_{n+3}, d_n, k_{n+2}), 0, 0),$$

as this calculation shows (note that d_n is the code for $DC_n(x, a, b, c, \sigma)$, $n = \mathrm{lh}(\sigma)$, which for $n > 0$ can be obtained from DC by a suitable application of $\mathbf{P}_{n,j}^m$)

$$\begin{aligned} \{i_n\}_\Theta^{n+1}(y, \sigma) &= \{p_{n+3,2,0}\}_\Theta^{n+4}(S_{n+3}^2(c_{n+3}, d_n, k_{n+2}), 0, 0, y, \sigma) \\ &= \{c_{n+3}\}_\Theta^{n+5}(d_n, k_{n+2}, y, 0, 0, \sigma) \\ &= \{d_n\}_\Theta^{n+4}(\{k_{n+2}\}_\Theta^{n+3}(y, 0, 0, \sigma), y, 0, 0, \sigma) \\ &= \begin{cases} y, & \text{if } y \in C \\ 1, & \text{if } y \notin C. \end{cases} \end{aligned}$$

1.2.4 Constant Functions on C. The constant functions can be introduced in different ways. For $y \in C$ we see that

$$DC(x, y, y, 0) = y,$$

for all x. We also see that for $y \in C$

$$\{S_n^1(i_n, y)\}_\Theta^n(\sigma) = \{i_n\}_\Theta^{n+1}(y, \sigma) = y,$$

for all sequences σ.

1.2.5 Ordered Pair on C. The existence for arbitrary precomputation theories of an ordered pair function on C with inverses, follows from the axioms. First choose a code $e \in C$ such that

$$\{e\}(a, b, c, x) = DC(x, a, b, c).$$

We omit the super- and subscript on $\{a\}_\Theta^n$ when the meaning is clear from the context. Define for $a, b \in C$:

(i) $\quad M(a, b) = S_2^2(e, a, b).$

Next we define for $z \in C$ the "inverses"

(ii) $\quad K(z) = \{z\}(e, e)$

(iii) $\quad L(z) = \{z\}(e, e')$, where $e' \neq e, e' \in C$.

More accurately we should define K and L by combining DC and composition, e.g.

$$K(z) = \begin{cases} \{z\}(e, e) & \text{if } z \in C \\ 1 & \text{if } z \notin C, \end{cases}$$

where $\{z\}(e, e)$ as a function of z can be obtained using $\mathbf{P}_{n,j}^m$ and S_n^m in a suitable way (the enumeration property of $\mathbf{P}$).

M as a map from $C \times C$ to C is 1-1, since $M(a, b) = M(a', b')$ implies that

$$\{S_2^2(e, a, b)\}(x, y) = \{S_2^2(e, a', b')\}(x, y),$$

for all x, y. Hence $a = a'$ and $b = b'$. Further we see that

$$K(M(a, b)) = \{S_2^2(e, a, b)\}(e, e) = \{e\}(a, b, e, e) = a,$$
$$L(M(a, b)) = \{S_2^2(e, a, b)\}(e, e') = \{e\}(a, b, e, e') = b.$$

Thus we have ordered pair and inverses, although they may not be the most natural choices in a specifically given precomputation theory. In particular, if

$A = C$, the pairing function M will be defined for all $a, b \in A$ with the correct properties.

1.2.6 The Fixed-point Theorem. We conclude our list of elementary examples by adapting the usual proof of the fixed-point theorem to arbitrary computation theories.

Theorem. *Let Θ be a precomputation theory on $\mathfrak{A}$. For every $n + 1$-ary Θ-computable* pmv *function f there exists an $a \in C$ such that for all σ from A*

$$\{a\}_\Theta^n(\sigma) = f(a, \sigma).$$

Proof. Define f_1 and S with Θ-codes $\hat{f}_1$, $\hat{S}$, respectively such that

$$\begin{aligned} f_1(x, y, \sigma) &= f(x, \sigma) \\ S(y, \sigma) &= S_1^1(y, y), \quad y \in C. \end{aligned}$$

Let

$$a = S_1^1(S_{n+1}^2(c_{n+1}, \hat{f}_1, \hat{S}), S_{n+1}^2(c_{n+1}, \hat{f}_1, \hat{S})),$$

a simple calculation proves the theorem.

1.3 Semicomputable Relations

For the sake of completeness we include a short discussion of computable and semicomputable relations within the context of a general precomputation theory. This is a topic to which we will return at greater length in later parts of this work.

1.3.1 Definition. Let Θ be a precomputation theory over a domain $\mathfrak{A} = \langle A, C; 0, 1\rangle$.

(i) A relation $R(\sigma)$ is *Θ-semicomputable* if there is a Θ-computable function f such that

$$R(\sigma) \text{ iff } f(\sigma) \simeq 0.$$

(ii) A relation $R(\sigma)$ is *Θ-computable* if there is a Θ-computable mapping f such that

$$R(\sigma) \text{ iff } f(\sigma) = 0.$$

Not many results about Θ-semicomputable sets can be proved in this generality.

It is, however, possible to show that if $A = C$, then the relation $(a, \sigma, z) \in \Theta$ is Θ-semicomputable.

1.3.2 Example. Let Θ be a precomputation theory on a domain $\mathfrak{A}$ where $A = C$. We show that the relation

$$(a, \sigma, z) \in \Theta,$$

is Θ-semicomputable.

First define a function with code e such that

$$\{e\}(u, v, \sigma) = \begin{cases} 0 & \text{if } u = v \\ 1 & \text{if } u \neq v. \end{cases}$$

Next, let a^* be a Θ-code, computable from a, such that

$$\{a^*\}(z, \sigma) \simeq t \quad \text{iff} \quad \{a\}(\sigma) \simeq t.$$

We now observe that

$$\begin{aligned} C(\{e\}, \{a^*\}, z, \sigma) \simeq 0 \quad &\text{iff} \quad \{e\}(\{a^*\}(z, \sigma), z, \sigma) \simeq 0 \\ &\text{iff} \quad \{a\}(\sigma) \simeq z \\ &\text{iff} \quad (a, \sigma, z) \in \Theta. \end{aligned}$$

One way of obtaining a well-behaved theory is to add selection operators. Examples show that it is too restrictive to add a single-valued selection operator. Hence, we have here a case where multiple-valuedness could serve a real purpose. However, we shall in later parts of this study see greater advantages in retaining single-valuedness of Θ and use other means to obtain a good theory for the computable and semicomputable relations.

1.3.3 Definition. Let Θ be a precomputation theory over a domain $\mathfrak{A} = \langle A, C; 0, 1\rangle$. An *n-ary selection operator* for Θ is an $n + 1$-ary Θ-computable pmv function $q(a, \sigma)$, with Θ-code $\hat{q}$, such that:

> If there is an x such that $\{a\}_\Theta^{n+1}(x, \sigma) \simeq 0$, then $q(a, \sigma)\downarrow$, and for all x such that $q(a, \sigma) \simeq x$ we have $\{a\}_\Theta^{n+1}(x, \sigma) \simeq 0$.

In general $q(a, \sigma)$ may be a subset of the set of *all* x such that $\{a\}_\Theta^{n+1}(x, \sigma) \simeq 0$.

1.3.4 Theorem. *Let Θ be a precomputation theory over $\mathfrak{A}$ having selection operators.*

(i) *If $R(x, \sigma)$ is Θ-semicomputable, then so is $\exists x R(x, \sigma)$.*
(ii) *If $R(\sigma)$ and $S(\sigma)$ are Θ-semicomputable, then so is $R(\sigma) \vee S(\sigma)$.*
(iii) *R is Θ-computable iff R and $\neg R$ are Θ-semicomputable.*

We indicate the proofs: (i) Let r be a code such that $R(x, \sigma)$ iff $\{r\}(x, \sigma) \simeq 0$. Then $\exists x R(x, \sigma)$ iff $\{r\}(q(r, \sigma), \sigma) \simeq 0$.

(ii) Let r and s be Θ-codes for R and S, respectively. Let $f(r, s)$ be a code, Θ-computable in r and s, for a Θ-computable mapping such that $\{f(r, s)\}(0) = r$ and $\{f(r, s)\}(x) = s$, $x \neq 0$. Then $R(\sigma) \vee S(\sigma)$ iff $\exists v[\{\{f(r, s)\}(v)\}(\sigma) \simeq 0]$.

(iii) Let r and s be codes for R and $\neg R$, respectively. Let $m(r, s)$ be a code (Θ-computable in r,s) for the following Θ-semicomputable relation

$$(\{r\}(\sigma) \simeq 0 \wedge t = 0) \vee (\{s\}(\sigma) \simeq 0 \wedge t = 1).$$

Using the selection operator we get a Θ-computable mapping $f(\sigma) = q(m(r, s), \sigma)$ such that

$$R(\sigma) \quad \text{iff} \quad f(\sigma) = 0.$$

So much for the general theory. We turn our attention to an investigation of the structure of an arbitrary precomputation theory. But first we look at theories over ω.

1.4 Computing Over the Integers

In this section we shall briefly consider precomputation theories over the integers, i.e. we assume that $A = C = \omega$, and that the designated elements "0" and "1" really are 0 and 1. We shall also restrict ourselves to precomputation theories in which the successor function $s(x) = x + 1$ is computable. We will show that such theories are closed under the μ-operator, the predecessor function, and primitive recursion.

1.4.1 The μ-Operator. Let $f(\sigma, y)$ be any Θ-computable function. Define via the Fixed-point Theorem 1.2.6 a function $h(\sigma, y)$ by the condition

$$h(\sigma, y) = \begin{cases} 0 & \text{if } f(\sigma, y) = 0 \\ h(\sigma, y + 1) + 1 & \text{if } f(\sigma, y) \neq 0. \end{cases}$$

We see that $\mu y[f(\sigma, y) = 0] = h(\sigma, 0)$.

1.4.2 The Predecessor Function. We would like to define $p(x)$ using the μ-operator as follows

$$p(x) = \begin{cases} 0 & \text{if } x = 0 \\ \mu y[y + 1 = x] & \text{if } x \neq 0. \end{cases}$$

But this does not exactly fit into the format of 1.4.1. We would have to replace the condition $y + 1 = x$ by e.g. $x \dot{-} (y + 1) = 0$. The trouble is that $a \dot{-} b$ is usually defined using the predecessor function. However, we get around this difficulty using the idea of the construction in 1.4.1. Define (by substitution into DC):

$$h(x, y) = \begin{cases} 0 & \text{if } s(y) = x \\ h(x, s(y)) + 1 & \text{if } s(y) \neq x \end{cases}$$

then $p(x) = h(x, 0) = \mu y[y + 1 = x]$.

1.4.3 Primitive Recursion. With the predecessor function at hand, primitive recursion follows by a simple application of the Fixed-point Theorem 1.2.6. Let $g(\sigma)$ and $h(x, y, \sigma)$ be two given Θ-computable functions. We define

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

Remark. We seem to be using DC in all the examples 1.4.1 to 1.4.3. But the careful reader will observe that DC is not quite enough. We have to use WDC, weak definition by cases, which by 2.7.4 is available.

It is now possible to state the following *minimality* result.

1.4.4 Theorem. *Let Θ be a precomputation theory over ω, and let f be a (Kleene) partial recursive function. Then f is Θ-computable.*

By the normal form theorem for partial recursive functions any such function can be represented in the form

$$f(\sigma) \simeq U(\mu y[g(\sigma, y) = 0]),$$

where g and U are primitive recursive functions. The proof now follows from the closure properties in 1.4.1 and 1.4.3.

Note that the theorem states an *extensional* result. We have not yet introduced a notion of "equivalence" or "extension" between theories. But granted a notion of extension $\Theta \leqslant H$ and granted that the set PR of partial recursive functions over ω is organized into a computation theory in some reasonable way, we would expect (and it is, indeed, true) that $PR \leqslant \Theta$, for all precomputation theories over ω.

1.4.5 Remark. We have so far assumed that the domain has the form $\mathfrak{A} = \langle A, C; 0, 1\rangle$. If we want to include the integers and the successor function, it is more natural to consider domains of the form $\mathfrak{A} = \langle A, C, N; s\rangle$, where $N \subseteq C \subseteq A$ and $\langle N, s \restriction N\rangle$ is (an isomorphic copy of) the integers, and s is defined as

(i) $$s(x) = \begin{cases} x + 1 & \text{if } x \in N \\ 0 & \text{if } x \notin N. \end{cases}$$

Then the Θ-computability of $s(x)$ implies that the characteristic function of N is Θ-computable.

If Θ is a theory over the more general type of domain $\mathfrak{A} = \langle A, C; 0, 1\rangle$, it is possible to construct a *successor* set, but this set will in general only be Θ-semicomputable and not Θ-computable.

We use the fact 1.2.5 that we have pairing on the code set C. First define a successor function

$$x' = M(0, x).$$

This defines x' for $x \in C$—extend to all of A by some suitable convention.

Next define

$$\begin{aligned}
&\mathbf{0} = M(1, 0)\\
&\mathbf{1} = \mathbf{0}' = M(0, M(1, 0))\\
&\mathbf{2} = \mathbf{1}' = M(0, M(0, M(1, 0)))\\
&\ldots
\end{aligned}$$

It is easy to verify that $\mathbf{0} \neq \mathbf{1}$, $\mathbf{1} \neq \mathbf{2}$, etc. Externally we have constructed a successor set $\mathbf{N} = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots\}$, and $\langle \mathbf{N}, '\rangle$ is isomorphic to the integers and the successor function. But we cannot, in general, restrict $s(x)$ as in (i) above and still preserve its Θ-computability, $\mathbf{N}$ need not be Θ-computable.

Expanding on this possibility we may make a contact with the theory of non-standard models and non-transitive admissible sets and see how they, in fact, fall under the scope of our axioms.

However, our main interest is towards the "hard core" of recursion theory, and we shall freely assume enough coding apparatus to make life easy—even if it is not always strictly necessary (see e.g. Definition 1.5.1 and the following Remark 1.5.2). But if the reader should insist on the extreme generality of Section 1.2 we recommend that he looks at Moldestad [106] to appreciate what pathologies then may obtain.

1.5 Inductively Defined Theories

We shall now retreat a bit from the, perhaps, too great generality of Sections 1.1 to 1.3. From now on we shall assume that our precomputation theories are *singlevalued*, i.e.

$$\text{if} \quad (a, \sigma, z) \in \Theta \quad \text{and} \quad (a, \sigma, w) \in \Theta, \quad \text{then} \quad z = w.$$

We shall further assume that a copy $\langle N, s\restriction N\rangle$ of the integers is included in the computation domain, and that the domain is provided with a coding scheme

$\langle M, K, L\rangle$. Definition 1.1.1 will thus be revised as follows:

1.5.1 Definition. A *computation domain* is a structure

$$\mathfrak{A} = \langle A, C, N; s, M, K, L\rangle,$$

where $N \subseteq C \subseteq A$ and $\langle N, s \restriction N\rangle$ is (an isomorphic copy of) the integers with s as *successor function* (defined as in 1.4.5).

$\langle M, K, L\rangle$ is a *pairing structure* on $\mathfrak{A}$, i.e.

(1) M is a pairing function on A, i.e. M is total and $M(a, b) = M(a_1, b_1)$ implies $a = a_1$ and $b = b_1$.

(2) K and L are inverses to M, i.e.
$K(M(a, b)) = a$ and $L(M(a, b)) = b$.

We extend the notational conventions following Definition 1.1.1 using $m, n, k, l, i, j, \ldots$ for elements in N.

1.5.2 Remarks. We shall see that we need a pairing structure on all of A in order to construct a universal function for a given precomputation theory (see Section 1.6.2). For many purposes we need only assume a C-restricted pairing structure, i.e. M is an ordered pair on C with K and L as inverses on C. For the general theory we could get away with a domain of the form $\mathfrak{A} = \langle A, C, N; s\rangle$, since a pairing structure on C exists by 1.2.5. However, as we shall note below, we would have to introduce some *external* pairing in constructing the theory $PR[\mathbf{f}]$, the precomputation theory "generated" by the functions $\mathbf{f}$. Hence, since we almost always have some coding scheme in mind over a given domain, we might as well include a pairing structure $\langle M, K, L\rangle$ as part of the data specifying the computation domain $\mathfrak{A}$.

The new definition of a computation domain necessitates certain changes in the definition of a precomputation theory. We must insist that the successor function s and the pairing structure $\langle M, K, L\rangle$ are Θ-computable. We shall also assume that the codes c_n, $p_{n,j,m}$, and $s_{n,m}$ (see Definition 1.1.10) are Θ-computable mappings of the parameters. This allows for more uniformity in defining computable functions and relations, and is quite essential in the subsequent theory.
The "new" Definition 1.1.10 is as follows:

1.5.3 Definition. Let Θ be a (single-valued) computation set on the domain $\mathfrak{A} = \langle A, C, N; s, M, K, L\rangle$. Θ is called a *precomputation theory* on $\mathfrak{A}$ if there exist Θ-computable mappings p_1, p_2, and p_3 such that

(i) for each n, j $(0 \leqslant j < n)$ and m the functions and functionals $s, M, K, L, DC, \mathbf{C}^n$, and $\mathbf{P}^m_{n,j}$ are Θ-computable with Θ-codes $\hat{s}, m, k, l, d, c_n = p_1(n)$, and $p_{n,j,m} = p_2(j, n, m)$ respectively;

(ii) Θ satisfies the iteration property (1.1.9), i.e. for each $n, m \geqslant 0$ $s_{n,m} = p_3(n, m)$ is a Θ-code for a mapping S^n_m such that for all $a \in C$, all sequences σ from C, and all τ from A,

$$\{a\}^{n+m}_{\Theta}(\sigma, \tau) = \{S^n_m(a, \sigma)\}^m_{\Theta}(\tau),$$

where $n = \operatorname{lh}(\sigma)$ and $m = \operatorname{lh}(\tau)$.

The pairing structure $\langle M, K, L\rangle$ will now be extended to a coding and decoding of arbitrary n-tuples. There are many ways of doing this. We do it by means of the following two auxiliary functions:

M^* is defined by iteration of M:

$$M^*(\;) = 1,$$
$$M^*(x_1, \ldots, x_{n+1}) = M(x_1, M^*(x_2, \ldots, x_{n+1})).$$

By primitive recursion we define functions L_i, $i \in N$:

$$L_1(x) = L(x),$$
$$L_{n+1}(x) = L(L_n(x)).$$

1.5.4 Definition. Ordered n-tuples and inverses are defined as follows:

A $\langle x_1, \ldots, x_n\rangle = M(n, M^*(x_1, \ldots, x_n))$.
B $\operatorname{lh}(x) = K(x)$.
C $(x)_i = K(L_i(x))$, if $i \neq \operatorname{lh}(x)$ or $\operatorname{lh}(x) \notin N$.
$(x)_i = L(L_i(x))$, if $i = \operatorname{lh}(x) \in N$.

1.5.5 Remark. By adapting the arguments of 1.4.2 we see that any precomputation theory Θ has a predecessor function on N. Hence, by 1.4.3, Θ will be closed under primitive recursion on N. Therefore the auxiliary functions M^* and L_i, $i \in N$, and the extended pairing structure of 1.5.4, are Θ-computable for every precomputation theory Θ.

With these preliminaries out of the way we shall, given any sequence

$$\mathbf{f} = f_1, \ldots, f_l,$$

of partial functions, construct a precomputation theory on $\mathfrak{A}$ *generated* by the given list $\mathbf{f}$.

1.5.6 Construction of $\Gamma_{\mathbf{f}}(\Theta)$. Let $\mathbf{f} = f_1, \ldots, f_l$ be a list of functions on A. $\Gamma_{\mathbf{f}}$ shall be a monotone operator acting on arbitrary sets of tuples (a, σ, z), where $\operatorname{lh}(a, \sigma, z) \geqslant 2$. The definition is by the following set of clauses:

1 $(\langle 1, 0\rangle, x, s(x)) \in \Gamma_{\mathbf{f}}(\Theta)$
2 $(\langle 2, 0\rangle, x, y, M(x, y)) \in \Gamma_{\mathbf{f}}(\Theta)$

3 $(\langle 3, 0\rangle, x, K(x)) \in \Gamma_{\mathbf{f}}(\Theta)$

4 $(\langle 4, 0\rangle, x, L(x)) \in \Gamma_{\mathbf{f}}(\Theta)$

5 $(\langle 5, 0\rangle, x, a, b, c, DC(x, a, b, c)) \in \Gamma_{\mathbf{f}}(\Theta)$

6 If $\exists u[(\hat{g}, \sigma, u) \in \Theta$ and $(\hat{f}, u, \sigma, z) \in \Theta]$, then $(\langle 6, 0\rangle, \hat{f}, \hat{g}, \sigma, z) \in \Gamma_{\mathbf{f}}(\Theta)$

7 Let $0 \leqslant j < n$ and τ any m-tuple from A, if $(\hat{f}, x_{j+1}, x_1, \ldots, x_j, x_{j+2}, \ldots, x_n, z) \in \Theta$, then $(\langle 7, j\rangle, \hat{f}, x_1, \ldots, x_n, \tau, z) \in \Gamma_{\mathbf{f}}(\Theta)$

8 If $a, x_1, \ldots, x_n \in C$ and $(a, x_1, \ldots, x_n, y_1, \ldots, y_m, z) \in \Theta$, then $(\langle 8, a, x_1, \ldots, x_n\rangle, y_1, \ldots, y_m, z) \in \Gamma_{\mathbf{f}}(\Theta)$

9 Let $\mathbf{f} = f_1, \ldots, f_l$ be the given list of functions. If $f_i(t_1, \ldots, t_{n_i}) \simeq z$, $1 \leqslant i \leqslant l$, then $(\langle 9, i\rangle, t_1, \ldots, t_{n_i}, z) \in \Gamma_{\mathbf{f}}(\Theta)$.

We note that clauses **1** to **5** introduces the basic functions; the successor function s, the pairing structure $\langle M, K, L\rangle$, and definition by cases DC. s and $\langle M, K, L\rangle$ are part of the data provided by the structure $\mathfrak{A}$. Clauses **6** and **7** introduces the functionals $\mathbf{C}^n$ and $\mathbf{P}^m_{n,j}$, respectively. Clause **8** introduces the iteration property, i.e. the functions S^n_m. Finally, clause **9** introduces the list **f**.

The operator $\Gamma_{\mathbf{f}}$ is monotone and has a least fixed-point $\Theta^*_{\mathbf{f}} = \bigcup \Theta^\xi_{\mathbf{f}}$, where $\Theta^\xi_{\mathbf{f}} = \Gamma_{\mathbf{f}}(\bigcup_{\eta<\xi} \Theta^\eta_{\mathbf{f}})$. In the simple setting of 1.5.6, $\Theta^*_{\mathbf{f}} = \Theta^\omega_{\mathbf{f}}$.

1.5.7 Definition. Let $\mathfrak{A}$ be a computation domain and $\mathbf{f} = f_1, \ldots, f_l$ a list of functions of A. The computation set generated by **f** over $\mathfrak{A}$, which will be called the *prime computation set* in **f**, is defined as the least fixed-point of $\Gamma_{\mathbf{f}}$ and denoted by PR[**f**], i.e.

$$\mathrm{PR}[\mathbf{f}] = \Theta^*_{\mathbf{f}} = \text{least fixed-point of } \Gamma_{\mathbf{f}},$$

Associated with the theory PR[**f**] are two notions which will play an important role in our subsequent investigations, *subcomputation* and *length of computation.*

1.5.8 Definition. Let PR[**f**] be a computation set as in Definition 1.5.7. For each $(a, \sigma, z) \in \mathrm{PR}[\mathbf{f}]$ we set

$$|a, \sigma, z|_{\mathrm{PR}[\mathbf{f}]} = \text{least } \xi \text{ such that } (a, \sigma, z) \in \Theta^\xi_{\mathbf{f}}.$$

$|a, \sigma, z|_{\mathrm{PR}[\mathbf{f}]}$ is called the *length of the computation* (a, σ, z). For a $\Gamma_{\mathbf{f}}$ as in 1.5.6 the length is a natural number.

1.5.9 Definition. For each $(a, \sigma, z) \in \mathrm{PR}[\mathbf{f}]$ we define the set of *immediate subcomputations.*

(i) If $a = \langle 1, 0\rangle, \langle 2, 0\rangle, \langle 3, 0\rangle, \langle 4, 0\rangle, \langle 5, 0\rangle$, or $\langle 9, 0\rangle$, then (a, σ, z) has no immediate subcomputations.

(ii) $(\langle 6, 0\rangle, \hat{f}, \hat{g}, \sigma, z)$ has $(\hat{g}, \sigma, u)$ and $(\hat{f}, u, \sigma, z)$ as immediate subcomputations. (Note u is uniquely determined.)

(iii) $(\langle 7, j\rangle, \hat{f}, x_1, \ldots, x_n, \tau, z)$ has $(\hat{f}, x_{j+1}, x_1, \ldots, x_j, x_{j+2}, \ldots, x_n, z)$ as immediate subcomputation.

(iv) $(\langle 8, a, x_1, \ldots, x_n\rangle, y_1, \ldots, y_m, z)$ has $(a, x_1, \ldots, x_n, y_1, \ldots, y_m, z)$ as immediate subcomputation.

The *subcomputation* relation is defined as the transitive closure of the immediate subcomputation relation. If (a, σ, z) is a subcomputation of (b, τ, w), we write $(a, \sigma, z) <_{\mathrm{PR[f]}} (b, \tau, w)$. The *subcomputation tree* of a $(a, \sigma, z) \in \mathrm{PR}[\mathbf{f}]$ is the set of subcomputations of (a, σ, z) with the relation $<_{\mathrm{PR[f]}}$.

We note that if $(a, \sigma, z) <_{\mathrm{PR[f]}} (b, \tau, w)$, then $|a, \sigma, z|_{\mathrm{PR[f]}} < |b, \tau, w|_{\mathrm{PR[f]}}$, but the converse is not necessarily true.

1.5.10 Proposition. *The prime computation set* PR[**f**] *is a precomputation theory on* $\mathfrak{A}$ *in which each function of the list* **f** *is computable.*

Proofs of facts such as the above proposition tend to be long, tedious, and entirely a matter of routine. They will therefore mostly be omitted. Since this is the first case and since there are a few details to be observed, we shall, however, this time indicate a few of the steps. To show that each f_i in **f** is PR[**f**]-computable, we must find a code $\hat{f}_i$ and verify that

$$\text{(i)} \qquad f_i(t_1, \ldots, t_{n_i}) \simeq z \quad \text{iff} \quad (\hat{f}_i, t_1, \ldots, t_{n_i}, z) \in \mathrm{PR}[\mathbf{f}].$$

The obvious choice for the code is $\hat{f}_i = \langle 9, i\rangle$. Since $\mathfrak{A}$ has a pairing structure, $\langle 9, i\rangle = M(2, M(9, M(i, 1))) \in C$. The implication from left to right in (i) is an immediate consequence of clause **9** in 1.5.6, the reverse implication must formally be proved by induction on the length of the computation $(\hat{f}, t_1, \ldots, t_{n_i}, z) \in \mathrm{PR}[\mathbf{f}]$.

In a similar way we verify that s, $\langle M, K, L\rangle$, DC, $\mathbf{C}^n$, and $\mathbf{P}^m_{n,j}$ are PR[**f**]-computable and that the codes $c_n = \langle 6, 0\rangle$ and $p_{n,j,m} = \langle 7, j\rangle$ are PR[**f**]-computable mappings of the parameters, the latter is, of course (!), routine to do. The reader is invited to verify that

$$\hat{p}_1 = \langle 8, \langle 7, 3\rangle, \langle 5, 0\rangle, \langle 6, 0\rangle, \langle 6, 0\rangle, 0\rangle,$$

is an appropriate choice of code for the mapping $p_1(n) = \langle 6, 0\rangle$.

This leaves the iteration property: We have to construct a code $\hat{p}_3$ for a mapping $p_3(n, m)$ which itself shall be a code in the theory PR[**f**] for the mapping

$$S^n_m(a, x_1, \ldots, x_n) = \langle 8, a, x_1, \ldots, x_n\rangle.$$

For this we need the extended coding structure 1.5.4. So our task will be to construct codes for various functions there needed. We do this by first verifying that the *set* PR[**f**] has the fixed-point property 1.2.6. Next we construct in PR[**f**] a code for the predecessor function on N, 1.4.2. Finally we adapt the argument of 1.4.3 to show that the *set* PR[**f**] is closed under primitive recursion on N. With this at hand we can, granted the necessary strength of will, construct codes in PR[**f**] for the functions in 1.5.4, from which we easily obtain the code $\hat{p}_3$.

1.5.11 Remarks. (1) The reader should observe that we have repeatedly *used* the

S^n_m-function (*via* clause **8** of 1.5.6) in order to prove that it has an index computable in the theory. (2) We refer back to Remark 1.5.2: It would have been possible to introduce the clauses **1**–**7** and **9** of 1.5.6 by choosing appropriate elements of the code set C but, without having some sort of pairing mechanism at disposal, it is difficult to see how to introduce the S^n_m-function such that: (i) the code for S^n_m is a computable function of n, m, and (ii) such that the iteration property

$$\{S^n_m(a, x_1, \ldots, x_n)\}(y_1, \ldots, y_m) = \{a\}(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

will obtain. This is the reason for including the pairing structure $\langle M, K, L\rangle$ in Definition 1.5.1.

1.6 A Simple Representation Theorem

We now turn to a converse proposition 1.5.8: Every precomputation theory on $\mathfrak{A}$ is of the form PR[**f**] for a suitable list **f**. In order to state the result precisely we need a notion of *equivalence* for precomputation theories.

1.6.1 Definition. Let Θ and H be two precomputation theories on the same domain $\mathfrak{A}$. We say that *H extends Θ*,

$$\Theta \leqslant H,$$

if there is an H-computable mapping $p(a, n)$ such that

$$(a, \sigma, z) \in \Theta \quad \text{iff} \quad (p(a, n), \sigma, z) \in H,$$

where $n = \mathrm{lh}(\sigma)$.

If $\Theta \leqslant H$ and $H \leqslant \Theta$, we say that Θ and H are *equivalent*, in symbols

$$\Theta \sim H.$$

Equivalent theories have the same computable functions. What 1.6.1 adds is that we have a computable transformation on codes, i.e. if $\Theta \sim H$ and f is Θ-computable with code $\hat{f}_\Theta$, we can inside H compute a code $\hat{f}_H = p(\hat{f}_\Theta, n)$ for f as an H-computable function. It is usually this stronger property we establish when we prove the "equivalence" between given "computation" theories, e.g. the equivalence between Turing computability and μ-recursion.

1.6.2 Universal Function. Implicit in the Θ-computability of the functional $\mathbf{P}^m_{n,j}$ lies the fact that each precomputation theory has an enumerating function, i.e. for each n there is a Θ-computable function f_n such that

$$(a, \sigma, z) \in \Theta \quad \text{iff} \quad f_n(a, \sigma) \simeq z.$$

A code for f_n can be computed in Θ from n. It is therefore possible to code up the functions f_n, $n \in N$, inside Θ to get a Θ-computable function f such that

$$\begin{aligned}(a, \sigma, z) \in \Theta \quad &\text{iff} \quad f_n(a, \sigma) \simeq z \\ &\text{iff} \quad f(n, \langle a, \sigma\rangle) \simeq z.\end{aligned}$$

Construct the theory PR[f]; it is not difficult inside this theory to define a function $p(a, n)$ such that $f(n, \langle a, \sigma\rangle) \simeq z$ iff $\{p(a, n)\}(\sigma) \simeq z$.

We conclude *that every precomputation theory* Θ *is reducible to some theory* PR[f], *where f is a Θ-computable partial function.*

A converse seems at first obvious. If f is Θ-computable, surely PR[f] $\leqslant \Theta$. It is indeed true, but needs a careful proof. We have to analyze the computation procedure given by PR[f] and see that it can be carried out *inside* Θ.

1.6.3 Theorem: Simple Representation. *Let Θ be a precomputation theory on the domain $\mathfrak{A}$. There exists a Θ-computable function f such that $\Theta \sim$* PR[f].

"Simple" in Theorem 1.6.3 will stand in contrast to "faithful" in Theorem 2.7.3. See 2.7.5 for further comments.

To prove 1.6.3 it is sufficient to show that if f is any Θ-computable function, then PR[f] $\leqslant \Theta$. The proof will be divided into four steps.

A. Program for the Proof. Recall from 1.5.6 and 1.5.7 that PR[f] $= \bigcup_{n<\omega} \Gamma_n$, where $\Gamma_0 = \varnothing$ and $\Gamma_{n+1} = \Gamma_f(\Gamma_n)$. We will construct a Θ-computable (partial) function $r(n, \langle a, \sigma\rangle)$ such that if $r(n, \langle a, \sigma\rangle) \simeq 0$, then $(a, \sigma, z) \in \Gamma_n$, where z is the unique value of the computation $\{a\}^n_{\mathrm{PR}[f]}(\sigma)$. We shall further see to that if $r(n, \langle a, \sigma\rangle)\downarrow$, then $r(m, \langle a, \sigma\rangle)\downarrow$ for all $m < n$. Define

$$q(\langle a, \sigma\rangle) \simeq \mu n[r(n, \langle a, \sigma\rangle) \simeq 0].$$

Since the μ-operator on N is Θ-computable, q is also Θ-computable. We note that if $(a, \sigma, z) \in$ PR[f], then $q(\langle a, \sigma\rangle)$ gives the "first" stage of the inductive generation of PR[f] at which this can be decided.

Our next task will be to define a Θ-computable function $p(n, \langle a, \sigma\rangle)$ which, whenever $(a, \sigma, z) \in \Gamma_n$, calculates z from n and the pair $\langle a, \sigma\rangle$.

Combining p and q shows that

$$\begin{aligned}(a, \sigma, z) \in \mathrm{PR}[f] \quad &\text{iff} \quad p(q(\langle a, \sigma\rangle), \langle a, \sigma\rangle) \simeq z \\ &\text{iff} \quad (t(a, n), \sigma, z) \in \Theta,\end{aligned}$$

where $t(a, n)$ is easily constructed from the line above. We see that $t(a, n)$ is Θ-computable and PR[f] $\leqslant \Theta$ *via* this t.

B. How to Compute Inside PR[f]. To prepare for the construction of r, p and q we illustrate how to decide in Γ_6 if there is some z such that $(\langle 6, 0\rangle, \sigma, z) \in \Gamma_6$. First of all σ must be of the form $\sigma = \hat{f}, \hat{g}, \sigma_1$, where $\hat{f}, \hat{g} \in C$ (this can be decided in Θ). What we so far have been given is an *incomplete* tuple $(\langle 6, 0\rangle, \hat{f}, \hat{g}, \sigma_1, \Box_1)$, where $\Box_1$ is a "blank" to be filled in by some z, if there is indeed some computation $(\langle 6, 0\rangle, \sigma, z)$ which has been put into $\mathrm{PR}[f]$ at stage 6, i.e. into Γ_6. Our search may look like the following diagram; i.e. we are trying to construct the subcomputation tree of $(\langle 6, 0\rangle, \hat{f}, \hat{g}, \sigma_1, \Box_1)$:

$$\begin{tikzcd}
 & \Gamma_6 : (\langle 6, 0\rangle, \hat{f}, \hat{g}, \sigma_1, \Box_1) \arrow[dl] \arrow[dr] & & \\
\Gamma_5 : (\hat{f}, \Box_2, \sigma_1, \Box_1) \arrow[d, dashed] & & \Gamma_5 : (\hat{g}, \sigma_1, \Box_2) \arrow[dl] & \\
{} & \Gamma_4 : (\hat{g}_1, \sigma_2, \Box_2) \arrow[dl] \arrow[dr] & & \\
\Gamma_3 : (\hat{h}_1, \Box_3, \sigma_3, \Box_2) \arrow[d, dashed] & & \Gamma_3 : (\hat{g}_2, \sigma_3, \Box_3) \arrow[dl] \arrow[dr] & \\
{} & \Gamma_2 : (\hat{h}_2, \Box_4, \sigma_4, \Box_3) \arrow[d, dashed] & & \Gamma_2 : (\hat{g}_3, \sigma_4, \Box_4) \arrow[dl] \\
 & {} & \Gamma_1 : (\hat{g}_4, \sigma_5, \Box_4) &
\end{tikzcd}$$

At stage Γ_6 we have the incomplete tuple $(\langle 6, 0\rangle, \hat{f}, \hat{g}, \sigma_1, \Box_1)$. There must be two *immediate subcomputations* $(\hat{g}, \sigma_1, \Box_2)$, where the blank $\Box_1$ has disappeared, but a new blank $\Box_2$ is introduced, and the "doubly indeterminate" tuple $(\hat{f}, \Box_2, \sigma_1, \Box_1)$. We first try to decide $(\hat{g}, \sigma_1, \Box_2)$. Suppose that it has one immediate predecessor, i.e. $\hat{g}$ must be of the form $\langle 7, -\rangle$ or $\langle 8, -\rangle$. Then we can uniquely read off what we have to decide at stage Γ_4, say $(\hat{g}_1, \sigma_2, \Box_2)$. Note that the blank $\Box_2$ is simply carried on. The code $\hat{g}_1$ may again indicate a substitution, which means that at stage Γ_3 we must investigate tuples $(\hat{h}_1, \Box_3, \sigma_3, \Box_2)$ and $(\hat{g}_2, \sigma_3, \Box_3)$. In the latter tuple $\Box_2$ has disappeared, but a new blank $\Box_3$ is introduced. This may again be a substitution, which leads us to investigate tuples $(\hat{h}_2, \Box_4, \sigma_4, \Box_3)$ and $(\hat{g}_3, \sigma_4, \Box_4)$ at stage Γ_2. Suppose that $\hat{g}_3$ is of the form $\langle 7, -\rangle$ or $\langle 8, -\rangle$. Then we must decide some $(\hat{g}_4, \sigma_5, \Box_4)$ at stage Γ_1. Note that everything in the diagram except the blanks is uniquely determined by the given data $\langle 6, 0\rangle$, σ.

Now we can start working backwards: $\hat{g}_4$ must be $\langle 1, 0\rangle$, $\langle 2, 0\rangle$, $\langle 3, 0\rangle$, $\langle 4, 0\rangle$, $\langle 5, 0\rangle$, or $\langle 9, 0\rangle$. Otherwise we can give the answer false to the original question. If $\hat{g}_4$ is either $\langle 1, 0\rangle$, $\langle 2, 0\rangle$, $\langle 3, 0\rangle$, $\langle 4, 0\rangle$, or $\langle 5, 0\rangle$, we can immediately fill in the blank $\Box_4$, as being the value of either s, M, K, L, or DC. If $\hat{g}_4 = \langle 9, 0\rangle$, we must ask Θ to supply the value for $\Box_4$, if such exists, i.e. if σ_5 has the appropriate length and $f(\sigma_5)\downarrow$.

Granted success at stage Γ_1, we move back to Γ_2 and fill in the blank $\Box_4$. Then we must try to decide $(\hat{h}_2, \Box_4, \sigma_4, \Box_3)$. If we succeed in this, we move back to Γ_3 and fill in $\Box_3$. Then we must attempt $(\hat{h}_1, \Box_3, \sigma_3, \Box_2)$. Again the process either succeeds, or we get an answer no, or the process is undefined by asking an inappropriate question about f. But note that the possible undefinabilities turn up in a way uniquely determined by the given data $\langle 6, 0\rangle$, σ. So if an undefinability turns up in deciding $(\langle 6, 0\rangle, \sigma, \Box_1)$ at stage Γ_n, then it turns up in every later attempt to decide $(\langle 6, 0\rangle, \sigma, \Box_1)$, i.e. at every stage Γ_m, $m \geqslant n$.

We now see how to work our way backwards filling in the blanks. And the following conclusion emerges: Either we get an answer YES, i.e. we are able to complete the computation $\{\langle 6, 0\rangle\}(\sigma)$ at stage Γ_6, or we get NO, in which case we may be able to complete $\{\langle 6, 0\rangle\}(\sigma)$ at some later stage, or we are blocked by UNDEFINED, and then the process will not be defined at any later stage.

If we get the answer YES, the process will provide us with the unique z such that $(\langle 6, 0\rangle, \sigma, z) \in \Gamma_6$. And we will then have succeeded in constructing the full subcomputation tree.

This completes our description of how to compute inside PR[f]. It remains to give the formal definitions of p, q, and r.

C. Definition of the Function $p(n, \langle a, \sigma\rangle)$. The function will be defined by induction on n. Since $\Gamma_0 = \varnothing$, we start with $n = 1$.

$n = 1$. In this case we set

$$\begin{aligned}
&p(0, \langle\langle 1, 0\rangle, x\rangle = s(x)\\
&p(0, \langle\langle 2, 0\rangle, x, y\rangle) = M(x, y)\\
&p(0, \langle\langle 3, 0\rangle, x\rangle) = K(x)\\
&p(0, \langle\langle 4, 0\rangle, x\rangle) = L(x)\\
&p(0, \langle\langle 5, 0\rangle, x, a, b, c\rangle) = DC(x, a, b, c)\\
&p(0, \langle\langle 9, 0\rangle, \sigma\rangle \simeq f(\sigma).
\end{aligned}$$

In all other cases p is undefined.

$n + 1, n \geqslant 1$. In all cases other than the ones treated below p is undefined.

$a = \langle 1, 0\rangle, \langle 2, 0\rangle, \langle 3, 0\rangle, \langle 4, 0\rangle, \langle 5, 0\rangle$, or $\langle 9, 0\rangle$: Define $p(n + 1, \langle a, \sigma\rangle)$ as in case $n = 1$.

$a = \langle 7, j\rangle$ or $\langle 8, a', x_1, \ldots, x_n\rangle$: Here we have one immediate subcomputation and the value of $p(n + 1, -)$ can be referred back to the value of $p(n, -)$:

$p(n + 1, \langle\langle 7, j\rangle, \hat{f}, \sigma, \tau\rangle) \simeq p(n, \langle \hat{f}, \sigma^j\rangle)$

$p(n + 1, \langle\langle 8, a', x_1, \ldots, x_n\rangle, \tau\rangle) \simeq p(n, \langle a', x_1, \ldots, x_n, \tau\rangle)$.

$a = \langle 6, 0\rangle$: Let the given data be $\langle 6, 0\rangle, \hat{f}, \hat{g}, \sigma$. We set

$p(n + 1, \langle\langle 6, 0\rangle, \hat{f}, \hat{g}, \sigma\rangle) \simeq p(n, \langle \hat{f}, p(n, \langle \hat{g}, \sigma\rangle), \sigma\rangle)$.

We see that we have a well-defined computation procedure in Θ. And we prove, by induction on n, that $(a, \sigma, z) \in \Gamma_n$ iff $p(n, \langle a, \sigma\rangle)\downarrow$ and $p(n, \langle a, \sigma\rangle) \simeq z$.

D. Definition of $r(n, \langle a, \sigma\rangle)$. The definition of $r(n, \langle a, \sigma\rangle)$ is almost identical to the definition of $p(n, \langle a, \sigma\rangle)$. There is, however, one crucial difference. In B we were led to three possibilities: Given $\langle a, \sigma\rangle$ and a stage Γ_n we either got the answer YES, the answer NO, or the answer UNDEFINED. In C we left $p(n, \langle a, \sigma\rangle)$ undefined if the answer was NO. In the case of $r(n, \langle a, \sigma\rangle)$ we give the value 1 in this case. And we set $r(n, \langle a, \sigma\rangle) = 0$ if the answer is YES. In this way we not only get that $(a, \sigma, z) \in \Gamma_n$ for some (and hence unique) z iff $r(n, \langle a, \sigma\rangle) \simeq 0$; but we also ensure that if $r(n, \langle a, \sigma\rangle) \simeq 0$, then $r(m, \langle a, \sigma\rangle)\downarrow$ for all $m < n$.

This completes the proof of Theorem 1.6.3.

1.7 The First Recursion Theorem

The aim of this section is to prove a version of the first fixed-point theorem for arbitrary precomputation theories. Our strategy will be to use the simple representation Theorem 1.6.3, viz. that any theory Θ is equivalent to a theory of the form PR$[g]$, where g is some partial function on the domain.

First we have some invariance questions to settle:

1.7.1 Proposition. (i) *If* PR$[g] \leqslant$ PR$[h]$ *and* f *is* PR$[h]$*-computable, then* PR$[g,f] \leqslant$ PR$[h]$.

(ii) PR$[f] \leqslant$ PR$[f, h]$.

(iii) *If* PR$[g] \leqslant$ PR$[h]$, *then* PR$[g,f] \leqslant$ PR$[h,f]$.

(i) and (ii) follows immediately from what we proved in 1.6.3, A–D, i.e. if a function f is Θ-computable, then PR$[f] \leqslant \Theta$. (i) now follows since PR$[g] \leqslant$ PR$[h]$ implies that g is PR$[h]$-computable; (ii) is equally obvious. (iii) is a corollary of (i) and (ii): g and f will both be PR$[h,f]$-computable, hence PR$[g,f] \leqslant$ PR$[h,f]$.

1.7.2 Remark. In proving Theorems 1.7.8 and 1.7.9 we need more detailed information about "subcomputations", which easily follows from an analysis of the arguments in 1.6.3, A–D. If f is PR$[g]$-computable with code $\hat{f}$, 1.7.1 tells us that PR$[g,f] \leqslant$ PR$[g]$ *via* some PR$[g]$-computable mapping $p(a, n)$. From 1.6.3 A–D it follows that we can safely assume that $p(\langle g, 0\rangle, n) = \hat{f}$, $\langle g, 0\rangle$ being the PR$[g,f]$-code for f.

We further note that the reduction can be arranged so that *subcomputations* are preserved. This needs some explanation: If $(a, \sigma, z) \in$ PR$[\mathbf{h}]$, for some functions $\mathbf{h}$, we construct the *tree of subcomputations* of (a, σ, z), which is uniquely determined from the tuple (a, σ, z). A tuple (b, τ, w) is a subcomputation of (a, σ, z) if it occurs as a node in this tree (see Definitions 1.5.8 and 1.5.9).

That subcomputations are preserved now means that if (b, τ, w) is a subcomputation of (a, σ, z) in PR$[g,f]$, then $(p(b, n'), \tau, w)$ is a subcomputation of $(p(a, n), \sigma, z)$ in PR$[g]$. In particular, $|p(b, n'), \tau, w|_{\mathrm{PR}[g]} < |p(a, n), \sigma, z|_{\mathrm{PR}[g]}$.

This is a common feature of reductions. One theory is reduced to or imbedded into another theory by imitating step-by-step the computations of the first inside the second. We shall in a more systematic way return to this in the next section on computation theories.

1.7.3 Definition. Let Θ be a precomputation theory on $\mathfrak{A}$, and assume by 1.6.3 that $\Theta \sim \mathrm{PR}[g]$, for some suitable partial function g. Let f be any function, we set

$$\Theta[f] = \mathrm{PR}[g, f].$$

By 1.7.1 $\Theta[f]$ is unique up to equivalence, i.e. if $\Theta \sim \mathrm{PR}[g]$ and $\Theta \sim \mathrm{PR}[h]$, then $\mathrm{PR}[g, f] \sim \mathrm{PR}[h, f]$.

1.7.4 Definition. Let Θ be a precomputation theory on $\mathfrak{A}$. A functional φ is called *strongly Θ-computable* if there exists a code $\hat{\varphi}$ such that the function $\lambda x \cdot \varphi(f, x)$ is $\Theta[f]$-computable with code $\hat{\varphi}$ for all f.

To remove any ambiguity let us agree on the following convention: $\Theta[f]$ is some theory $\mathrm{PR}[g, f]$, where g is such that $\Theta \sim \mathrm{PR}[g]$. We assume that in the construction of $\mathrm{PR}[g, f]$ f has code $\langle 9, 0\rangle$ and g has code $\langle 9, 1\rangle$, see 1.5.6.

1.7.5 Remarks. We observe the following invariance properties:

(i) Let $\Theta \sim H$: f is Θ-computable iff f is H-computable.
(ii) Let $\Theta \sim H$: φ is strongly Θ-computable iff φ is strongly H-computable.

Here (i) is immediate from the definition. (ii) follows from the following fact: Assume that $\Theta \sim \mathrm{PR}[g]$ and $H \sim \mathrm{PR}[h]$ and that $\Theta \leqslant H$, i.e. $\mathrm{PR}[g] \leqslant \mathrm{PR}[h]$. Hence $\mathrm{PR}[g, f] \leqslant \mathrm{PR}[h, f]$ by some mapping p which is independent of the particular function f, since f has the same "label" in each theory $\mathrm{PR}[g, f]$. But then any strongly Θ-computable φ is also strongly H-computable.

From 1.7.5 we can always assume that Θ has the form $\mathrm{PR}[g]$. Before turning to the fixed-point theorem we shall make a remark on strong versus weak computability.

1.7.6 Proposition. *If φ is strongly* $\mathrm{PR}[g]$*-computable, then φ is weakly* $\mathrm{PR}[g]$*-computable.*

Let $\hat{\varphi}$ be a code for φ as a strongly $\mathrm{PR}[g]$-computable functional, and let f be $\mathrm{PR}[g]$-computable with code $\hat{f}$. From 1.7.1 it follows that $\mathrm{PR}[g, f] \leqslant \mathrm{PR}[g]$ *via* a procedure which is a $\mathrm{PR}[g]$-computable function of $\hat{f}$. So from $\hat{\varphi}$ and $\hat{f}$ we can in $\mathrm{PR}[g]$ compute a code $p(\hat{\varphi}, \hat{f})$ for the function $\lambda x \cdot \varphi(f, x)$. By usual code manipulations we can find a code $\hat{\varphi}_1$ (as a function of $\hat{\varphi}$) such that $\{\hat{\varphi}_1\}(\hat{f}, x) \simeq \{p(\hat{\varphi}, \hat{f})\}(x)$. $\hat{\varphi}_1$ will then serve as a $\mathrm{PR}[g]$-code for φ as a weakly computable functional.

1.7.7 Example. Weak does not imply strong. This follows from simple cardinality considerations. Let Θ be ORT (= ordinary recursion theory over ω). There are countably many strongly computable ORT-functionals, since the code set is countable. But there are uncountably many weakly Θ-computable functionals: Weak computability is a requirement concerning the computable functions, outside this set the functional can behave exactly as it pleases.

We return to the main topic of this section.

1.7.8 Theorem. *Every consistent strongly* PR$[g]$*-computable functional* φ *has a least fixed-point, i.e. there exists a* f^* *such that*

(i) $\varphi(f^*, x) = f^*(x)$,
(ii) $\varphi(f, x) = f(x)$ *implies that* $f^* \subseteq f$.

The proof is rather standard: Let $f_0 = \varnothing$, $f_{n+1}(x) = \varphi(f_n, x)$, and set

$$f^*(x) = \lim f_n(x).$$

Since φ is strongly PR$[g]$-computable, φ is consistent, i.e. if $f \subseteq h$ and $\varphi(f, x)\downarrow$, then $\varphi(h, x)\downarrow$ and $\varphi(f, x) = \varphi(h, x)$. From this we see that $f_n \subseteq f_{n+1}$ for all n, hence f^* is well defined.

Proof of (i). First suppose that $f^*(x)\downarrow$, then $f^*(x) = f_{n+1}(x) = \varphi(f_n, x) = \varphi(f^*, x)$, by the consistency of φ. Conversely, suppose that $\varphi(f^*, x)\downarrow$ with value z. This means that $(\hat{\varphi}, x, z) \in \mathrm{PR}[g, f^*]$. f^* has by our convention code $\langle 9, 0\rangle$ in PR$[g, f^*]$. Let U be the set of pairs $\langle u, v\rangle$ such that $(\langle 9, 0\rangle, u, v)$ is a subcomputation of $(\hat{\varphi}, x, z)$. U is a finite set and there exists some f_n in the approximation to f^* such that $U \subseteq f_n$. It follows that $(\hat{\varphi}, x, z) \in \mathrm{PR}[g, f_n]$, i.e. $\varphi(f_n, x) = z$. We get $\varphi(f^*, x) = \varphi(f_n, x) = f_{n+1}(x) = f^*(x)$.

Proof of (ii). Let f be any fixed-point for φ, we prove by induction that $f_n \subseteq f$, for all n. Hence $f^* = \lim f_n \subseteq f$. For the induction step we assume that $f_{n+1}(x)\downarrow$. Then $f_{n+1}(x) = \varphi(f_n, x) = \varphi(f, x) = f(x)$, where the middle equality follows from the induction hypothesis.

1.7.9 Theorem. *Let* φ *be strongly* PR$[g]$*-computable. The least fixed-point* f^* *for* φ *is* PR$[g]$*-computable.*

Proof. In the proof of Proposition 1.7.6 we noted that if f is any PR$[g]$-computable function with code $\hat{f}$, there is a reduction procedure PR$[g, f] \leqslant$ PR$[g]$ which is a PR$[g]$-computable function of $\hat{f}$. This means, in particular, that there is a PR$[g]$-computable function p such that $p(\hat{\varphi}, \hat{f})$ is a PR$[g]$-code for the function $\lambda x \cdot \varphi(f, x)$, where $\hat{\varphi}$ is a code for φ as strongly PR$[g]$-computable.

We know that a least fixed-point f^* exists. It remains to construct a code for it as a PR$[g]$-computable function. The idea is simple, we want to construct a code $\widehat{f^*}$ such that $\{\widehat{f^*}\}_{\mathrm{PR}[g]} = \{p(\hat{\varphi}, \widehat{f^*})\}_{\mathrm{PR}[g]}(x)$. The right-hand side is here a PR$[g]$-

computable function $t(\widehat{f^*}, x)$. We can therefore use the Fixed-point Theorem 1.2.6 to find a code $\widehat{f^*}$ satisfying the equation and with the further property

(i) $\quad |p(\hat{\varphi}, \widehat{f^*}), x, z|_{\mathrm{PR}[g]} < |\widehat{f^*}, x, z|_{\mathrm{PR}[g]}$.

The code $\widehat{f^*}$ defines a PR[g]-computable function $\{\widehat{f^*}\}_{\mathrm{PR}[g]}$. We will show that it equals the least fixed-point f^* by verifying: (a) $\{\widehat{f^*}\}$ is a fixed-point for φ; (b) $\{\widehat{f^*}\} \subseteq f^*$.

(a) A simple calculation shows that $\{\widehat{f^*}\}$ is a fixed-point

$$\begin{aligned}\{\widehat{f^*}\}_{\mathrm{PR}[g]}(x) = z \quad &\text{iff} \quad \{p(\hat{\varphi}, \widehat{f^*})\}_{\mathrm{PR}[g]}(x) = z\\ &\text{iff} \quad \{\hat{\varphi}\}_{\mathrm{PR}[g, \{\widehat{f^*}\}]}(x) = z\\ &\text{iff} \quad \varphi(\{\widehat{f^*}\}, x) = z.\end{aligned}$$

(b) Suppose that $\{\widehat{f^*}\}(x) = z$, we must show that $f^*(x) = z$. This we do by induction assuming as induction hypothesis that this holds for all tuples $(\widehat{f^*}, u, v)$ such that

$$|\widehat{f^*}, u, v|_{\mathrm{PR}[g]} < |\widehat{f^*}, x, z|_{\mathrm{PR}[g]}.$$

Let $h \subseteq \{\widehat{f^*}\}$ be the subfunction needed in the given computation $\varphi(\{\widehat{f^*}\}, x) = z$ in PR[$g, \{\widehat{f^*}\}$]. We note that $\varphi(h, x) = z$. Consider now the computation $(p(\hat{\varphi}, \widehat{f^*}), x, z)$ in PR[g], we may conclude from Remark 1.7.2 that

(ii) $\quad |\widehat{f^*}, u, v|_{\mathrm{PR}[g]} < |p(\hat{\varphi}, \widehat{f^*}), x, z|_{\mathrm{PR}[g]}$,

for all pairs $\langle u, v\rangle \in h$.

This implies that $h \subseteq f^*$: If $h(u) = v$, then $\{\widehat{f^*}\}(u) = v$, and since $|\widehat{f^*}, u, v|_{\mathrm{PR}[g]} < |\widehat{f^*}, x, z|_{\mathrm{PR}[g]}$ (because of (i) and (ii)), the induction hypothesis tells us that $f^*(u) = v$.

From $\varphi(h, x) = z$ and $h \subseteq f^*$, the consistency of φ entails that $\varphi(f^*, x) = z$, i.e. $f^*(x) = z$. We conclude that $\{\widehat{f^*}\} \subseteq f^*$.

1.7.10 Remark. The reader will note that Theorems 1.7.8 and 1.7.9 with their proofs are in essence an adaptation of Kleene's presentation in *Introduction to Metamathematics* [78], §66. Our notion of strong Θ-computability corresponds to his notion of uniformity (see the discussion in §47 of *IMM*). In the next section on computation theories we will discuss a version of the first recursion theorem for weakly Θ-computable functionals.

1.7.11 Remark. Theorem 1.7.8 constructs the least fixed-point. The construction of Theorem 1.2.6 does not always lead to a "least" or "unique" solution. Consider the function

$$f(a, \sigma) = a,$$

in ORT over ω. Let a_0 be the fixed-point calculated following the proof of 1.2.6. Let a_1 be the fixed-point of $f'(a, \sigma) = 1 \cdot a$. We note that $a_0 \neq a_1$, that $\{a_0\}$ and $\{a_1\}$ are both total functions and fixed-points of f. But for no input σ is $\{a_0\}(\sigma) = \{a_1\}(\sigma)$.

If, however, a occurs in $f(a, \sigma)$ only through a part $\{a\}$, then the procedure of 1.2.6 is known to lead to the least fixed-point.

1.7.12 Remark. The material in this chapter is the basic core of any exposition of general recursion theory. The sources for the various combinatorial tricks are lost in the ancient history of recursion theory and the λ-calculus.

Our exposition is very much influenced by Kleene [83] and Moschovakis. [112, 113]. We should also mention the work of Strong [166], Wagner [169], and H. Friedman [33]. There is also an unpublished study by Aczel [2] on "enumeration systems" which contains the representation theorem 1.6.3 in the special case of domain ω. The thesis of L. Sasso [145] contains a great deal of material relevant for this part of the theory. In particular, he has the normal form theorem for a function recursive in a partial function.