Chapter 4

Finite Theories on Two Types

Moving up in types over the integers ω the notion of *finiteness* bifurcates. In higher types one must carefully distinguish between a *weak* and a *strong* notion (see Definition 3.1.2 and the connecting remarks). This was a phenomenon first observed by Y. Moschovakis [111] in his study of the hyperanalytic sets. He proved that the set of reals semicomputable (in the sense of Kleene) in ${}^3E$ is *not* closed under $\exists\alpha$, i.e. existential quantification over the reals. A further analysis reveals that higher type theories can be captured as theories on "two types", i.e. as computation theories on domains of the form $A = S \cup \mathrm{Tp}(S)$, where $\mathrm{Tp}(S) = \omega^S$, and where S is *strongly finite*, but $\mathrm{Tp}(S)$ is only *weakly finite*.

The distinction between two types can already be found in Moschovakis [114], viz. the notion of 2-point class. It was adopted by L. Harrington and D. MacQueen [55] in their proof of the Grilliot selection theorem. A general development of computation theories on two types was first presented by J. Moldestad [105]. His primary aim was to provide a "natural" setting for the general "plus-2" and "plus-1" theorems (see Chapter 7). But the theory can also serve as a framework for developing second-order definability results in general. A variant theory is the theory of Spector second-order classes (see Section 4.4 below).

4.1 Computation Theories on Two Types

We will study theories Θ on domains of the form $\mathfrak{A} = \langle A, S, \mathbf{S}\rangle$, where S is some infinite set, $A = S \cup \mathrm{Tp}(S)$, and $\mathbf{S}$ is a coding scheme for $\mathbf{S}$, i.e.

$$\mathbf{S} = \langle N, s, M, K, L\rangle,$$

where $N \subseteq S$ and $\langle N, s\rangle$ is isomorphic to the integers ω with the successor function. M, K, L are the usual pairing and projection functions on S. We assume that N is closed under M, K, and L. We also assume that $\mathrm{Tp}(S) = \omega^S$.

As usual we can introduce the functions and predicates associated with coding of finite tuples (see the discussion in Section 1.5). In particular we define a predicate $\mathrm{Seq}(r)$ by

$$\mathrm{Seq}(r) \quad \text{iff} \quad \exists n \exists r_1 \ldots \exists r_n [r = \langle r_1, \ldots, r_n\rangle].$$

In the concrete case of higher types we can "bound" the quantifiers in the definition of Seq(r). In the general case we must hypothesize the computability of $\mathbf{E}'_S$ so as to handle Seq(r).

Let * be the injection $S \to \mathrm{Tp}(S)$ defined by

$$r^*(s) = \begin{cases} 0 & \text{if } s = r \\ 1 & \text{if } s \neq r, \end{cases}$$

and $^-$ a function $A \to S$ defined by

$$x^- = \begin{cases} r & \text{if } x = r^* \\ 0 & \text{otherwise.} \end{cases}$$

With the help of these functions any coding scheme $\mathbf{S}$ for S can be lifted to A, e.g.

$$\begin{aligned} M(r, \alpha) &= \lambda s \cdot M(M(r^*(s), \alpha(s)), M(0, 1)) \\ M(\alpha, r) &= \lambda s \cdot M(M(\alpha(s), r^*(s)), M(1, 0)) \\ M(\alpha, \beta) &= \lambda s \cdot M(M(\alpha(s), \beta(s)), M(1, 1)), \end{aligned}$$

where $\alpha, \beta \in \mathrm{Tp}(S)$, $r, s \in S$.

$$\begin{aligned} K(\alpha) &= (\lambda s \cdot K(K(\alpha(s))))^- && \text{if } L(\alpha(0)) = M(0, 1), \\ &= \lambda s \cdot K(K(\alpha(s))) && \text{otherwise.} \\ L(\alpha) &= (\lambda s \cdot L(K(\alpha(s))))^- && \text{if } L(\alpha(0)) = M(1, 0), \\ &= \lambda s \cdot L(K(\alpha(s))) && \text{otherwise.} \end{aligned}$$

With these definitions the associated functions and predicates can be extended to A.

4.1.1 Definition. Let $\mathfrak{A} = \langle A, S, \mathbf{S} \rangle$ be a computation domain. The class PRF of *primitive recursive functions* is generated by the following schemes wherein σ is always a finite list of elements from A.

1. We have a base of nine initial functions

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

(ii) $$f(x, \sigma) = \begin{cases} 0 & \text{if } x \in S, \\ 1 & \text{if } x \notin S. \end{cases}$$

(iii) $$f(x, \sigma) = \begin{cases} x & \text{if } x \in S, \\ 0 & \text{if } x \notin S. \end{cases}$$

(iv) $$f(x, \sigma) = \begin{cases} x + 1 & \text{if } x \in N, \\ 0 & \text{if } x \notin N. \end{cases}$$

(v) $f(\sigma) = m \quad m \in N.$

(vi) $f(x, y, \sigma) = \begin{cases} M(x, y) & \text{if } x, y \in S, \\ 0 & \text{otherwise.} \end{cases}$

(vii) $f(x, \sigma) = \begin{cases} K(x) & \text{if } x \in S, \\ 0 & \text{otherwise.} \end{cases}$

(viii) $f(x, \sigma) = \begin{cases} L(x) & \text{if } x \in S, \\ 0 & \text{otherwise.} \end{cases}$

(ix) $f(x, y, \sigma) = \begin{cases} x(y) & \text{if } x \in \mathrm{Tp}(s), \quad y \in S, \\ 0 & \text{otherwise.} \end{cases}$

2. We then have closure under the following schemes:

(a) *Substitution:* $f(\sigma) = g(h(\sigma), \sigma)$.
(b) *Primitive recursion:* $f(0, \sigma) = g(\sigma)$
$f(n + 1, \sigma) = h(f(n, \sigma), n, \sigma), \quad n \in N$
$f(x, \sigma) = 0, \quad \text{if} \quad x \notin N.$
(c) *Permutation:* $f(\sigma) = g(\sigma')$, σ' permutation of σ.

Let $R_1, \ldots, R_k$ be predicates, $f_1, \ldots, f_l$ total functions with values in S, i.e. $f_i\colon A^{n_i} \to S$, and $F_1, \ldots, F_m$ total functionals, i.e. each F_i is total and has as arguments total functions with values in S.

We can extend Definition 4.1.1 in the following way.

4.1.2. Definition. Let **L** be a list $R_1, \ldots, R_k, f_1, \ldots, f_l, F_1, \ldots, F_m$ as specified above. The class of functions PRF(**L**) is obtained by adding the following initial schemes to Definition 4.4.1:

(x) $f_i(\sigma, \tau) = \begin{cases} 0 & \text{if } R_i(\sigma), \\ 1 & \text{if } \neg R_i(\sigma) \end{cases} \quad i = 1, \ldots, k.$

(xi) $f_i(\sigma, \tau) = f_i(\sigma) \quad i = 1, \ldots, l.$

(xii) $f_i(\sigma) = F_i(\lambda x \cdot g(x, \sigma)) \quad i = 1, \ldots, m.$

We also add the following closure scheme (allowing us to substitute a function for an element of Tp(S)).

(d) *Substitution:* $f(\sigma) = h(\lambda r \cdot g(r, \sigma), \sigma)$, where g takes values in N and r ranges over elements from S.

Note the following lemma.

4.1.3 Lemma. *The graphs of the functions* $*$, $^-$, *and of the functions associated with the extended coding scheme, and the extended predicate* Seq *are primitive recursive in the equality relation on S and the functional* $\mathbf{E}'_S$.

Note that $\mathbf{E}'_S$ is defined on total functions only, i.e.

$$\mathbf{E}'_S(f) = \begin{cases} 0 & \text{if } \exists s \in S.\ f(s) = 0 \\ 1 & \text{if } \forall s \in S.\ f(s) \neq 0, \end{cases}$$

where $f\colon A \to S$ is total. $\mathbf{E}_S$ is the extension of $\mathbf{E}'_S$ to partial functions.

We come now to the main notion of functions partial recursive in some list $\mathbf{L} = R_1, \ldots, R_k, \varphi_1, \ldots, \varphi_l, F_1, \ldots, F_m$, where $R_1, \ldots, R_k, F_1, \ldots, F_m$ are as above, but $\varphi_1, \ldots, \varphi_l$ are partial functions with values in S. There are 17 clauses in the definition of PR(**L**), which we give, with some reluctance, for the sake of completeness and explicitness.

4.1.4 Definition. Let $\mathfrak{A} = \langle A, S, \mathbf{S}\rangle$ be a computation domain and $\mathbf{L} = R_1, \ldots, R_k, \varphi_1, \ldots, \varphi_l, F_1, \ldots, F_m$ a list on $\mathfrak{A}$. The class PR(**L**) of functions partial recursive in **L** is given by the following inductive definition:

Let Γ be the following inductive operator

I $(\langle 1, n+1\rangle, x, \sigma, 0) \in \Gamma(X)$ if $x \in N$,
$(\langle 1, n+1\rangle, x, \sigma, 1) \in \Gamma(X)$ if $x \notin N$.

II $(\langle 2, n+1\rangle, x, \sigma, 0) \in \Gamma(X)$ if $x \in S$,
$(\langle 2, n+1\rangle, x, \sigma, 1) \in \Gamma(X)$ if $x \notin S$.

III $(\langle 3, n+1\rangle, x, \sigma, x) \in \Gamma(X)$ if $x \in S$,
$(\langle 3, n+1\rangle, x, \sigma, 0) \in \Gamma(X)$ if $x \notin S$.

IV $(\langle 4, n+1\rangle, x, \sigma, x+1) \in \Gamma(X)$ if $x \in N$,
$(\langle 4, n+1\rangle, x, \sigma, 0) \in \Gamma(X)$ if $x \notin N$.

V $(\langle 5, n, m\rangle, \sigma, m) \in \Gamma(X)$ $m \in N$.

VI $(\langle 6, n+2\rangle, x, y, \sigma, M(x, y)) \in \Gamma(X)$ if $x, y \in S$,
$(\langle 6, n+2\rangle, x, y, \sigma, 0) \in \Gamma(X)$ otherwise.

VII $(\langle 7, n+1\rangle, x, \sigma, K(x)) \in \Gamma(X)$ $x \in S$,
$(\langle 7, n+1\rangle, x, \sigma, 0) \in \Gamma(X)$ otherwise.

VIII $(\langle 8, n+1\rangle, x, \sigma, L(x)) \in \Gamma(X)$ $x \in S$,
$(\langle 8, n+1\rangle, x, \sigma, 0) \in \Gamma(X)$ otherwise.

IX $(\langle 9, n+2\rangle, x, y, \sigma, x(y)) \in \Gamma(X)$ $x \in \mathrm{Tp}(S)$, $y \in S$,
$(\langle 9, n+2\rangle, x, y, \sigma, 0) \in \Gamma(X)$ otherwise.

X If $\exists y[(e, \sigma, y) \in X \wedge (e', y, \sigma, x) \in X]$,
then $(\langle 10, n, e, e'\rangle, \sigma, x) \in \Gamma(X)$.

XI If $(e, \sigma, x) \in X$, then $(\langle 11, n+1, e, e'\rangle, 0, \sigma, x) \in \Gamma(X)$.
If $\exists y[(\langle 11, n+1, e, e'\rangle, m, \sigma, y) \in X \wedge (e', y, m, \sigma, x) \in X]$
then $(\langle 11, n+1, e, e'\rangle, m+1, \sigma, x) \in \Gamma(X)$.

XII If $(e, \sigma', x) \in X$, then $(\langle 12, n, e, i\rangle, \sigma, x) \in \Gamma(X)$,
where σ' is obtained from σ by moving the $i+1$-st object in σ to the front of the list.

XIII If $(e, \sigma, x) \in X$, then $(\langle 13, n+1\rangle, e, \sigma, x) \in \Gamma(X)$.

XIV $(\langle 14, j_i + n, i\rangle, \tau, \sigma, 0) \in \Gamma(X)$ if $R_i(\tau)$
$(\langle 14, j_i + n, i\rangle, \tau, \sigma, 1) \in \Gamma(X)$ if $\neg R_i(\tau)$,
where $\mathrm{lh}(\tau) = j_i$.

XV $(\langle 15, j_i + n, i\rangle, \tau, \sigma, \varphi_i(\tau)) \in \Gamma(X)$ if $\tau \in \mathrm{dom}(\varphi_i)$.

XVI If $\forall x \exists y (e, x, \sigma, y) \in X$, then $(\langle 16, n, e, i\rangle, \sigma, F_i(f)) \in \Gamma(X)$, where $f(x) = y$ iff $(e, x, \sigma, y) \in X$.

XVII If $\forall x \in S, \exists y \in N[(e, x, \sigma, y) \in X]$ and $(e', z, \sigma, u) \in X$, then $(\langle 17, n, e, e'\rangle, \sigma, u) \in \Gamma(X)$, where $z \in \mathrm{Tp}(S)$ is defined by $z(x) = y$ iff $(e, x, \sigma, y) \in X$.

We now let $\mathbf{PR}(\mathbf{L}) = \Gamma_\infty$ = the least fixed-point for Γ. $\mathbf{PR}(\mathbf{L})$ *is the set of functions which are partial recursive in* $\mathbf{L}$. By dropping XIII we get $\mathbf{PRF}(\mathbf{L})$, the functions primitive recursive in $\mathbf{L}$.

As usual we define for $(e, \sigma, x) \in \mathbf{PR}(\mathbf{L})$

$$|e, \sigma, x|_{\mathbf{L}} = \text{least } \xi \text{ such that } (e, \sigma, x) \in \Gamma^{\xi+1}.$$

Let $\kappa_{\mathbf{L}} = \sup\{|e, \sigma, x|_{\mathbf{L}} : (e, \sigma, x) \in \mathbf{PR}(\mathbf{L})\}$.

4.1.5 Remark. We see that clause XIII is the reflection scheme which leads from the primitive recursive to the partial recursive functions. As before, we note that the scheme XI for primitive recursion can be removed in presence of XIII.

The scheme XVII is an extended form for substitution, i.e. the scheme S8 of Kleene [83]. If $\lambda x \cdot \{e\}(x, \sigma)$ is total, then it is an element of $\mathrm{Tp}(S)$ and hence can occur as an argument for a function g. If this function g is computable, i.e. $g(\alpha, \sigma) = \{e'\}(\alpha, \sigma)$, this means that given α, σ by some oracle, we then can compute $\{e'\}(\alpha, \sigma)$. If α itself is not given by some oracle, but by a computation procedure, $\alpha = \lambda s \cdot \{e\}(s, \sigma)$, we should be able to compute g on α, σ, *not* by appealing to an oracle for both α, σ, but by using the computation procedure for α and the oracle only for σ, i.e.

$$f(\sigma) = g(\alpha, \sigma) = \{e'\}(\lambda s \cdot \{e\}(s, \sigma), \sigma),$$

should belong to $\mathbf{PR}(\mathbf{L})$ with an *index* e'' simply computable from e and e'. And this is precisely the content of scheme XVII.

Let us fix some further terminology. Our computations are single-valued, hence there is no loss of information in abbreviating $(e, \sigma, x) \in \mathbf{PR}(\mathbf{L})$ by $\langle e, \sigma\rangle$. In general we call (e, σ, x) a computation. It is *convergent* if it belongs to $\mathbf{PR}(\mathbf{L})$, we denote this by $\langle e, \sigma\rangle \downarrow$. Conversely, $\langle e, \sigma\rangle \uparrow$ means that for no x is $(e, \sigma, x) \in \mathbf{PR}(\mathbf{L})$, and we say that the computation $\langle e, \sigma\rangle$ *diverges*.

We note that given any pair $\langle e, \sigma\rangle$ we can start checking whether it is a convergent computation or not. If $\langle e, \sigma\rangle \downarrow$, then we can from Definition 4.1.4 define a notion of *immediate subcomputation* and *subcomputation* exactly as in Definition 1.5.9. If $\langle e, \sigma\rangle \uparrow$, then either e is not an index according to Definition 4.1.4, or σ is not of the form required by e, in this case we let $\langle e, \sigma\rangle$ be an immediate subcomputation of itself. If $\langle e, \sigma\rangle$ "looks" like a convergent computation, we can again define the immediate subcomputations of $\langle e, \sigma\rangle$—in this case at least one of them must be divergent.

In any case, for every $\langle e, \sigma\rangle$ there is a *well-defined notion of immediate subcomputation and of subcomputation.* And we have the following simple, but important theorem.

4.1.6 Theorem. *For all* $\langle e, \sigma\rangle$, $\langle e, \sigma\rangle\downarrow$ *iff the computation tree (i.e. the set of subcomputations) is well-founded.*

We shall state in the present framework the first and second recursion theorem. First some definitions. Let

$$\mathbf{F}(\varphi_1, \ldots, \varphi_k, \sigma),$$

be a partial functional with values in S, i.e. $\mathbf{F}$ is partial and defined on partial functions. $\mathbf{F}$ is *monotone* if whenever $\mathbf{F}(\varphi_1, \ldots, \varphi_k, \sigma)\downarrow$ and $\varphi_i \subseteq \psi_i$, $i = 1, \ldots, k$, then $\mathbf{F}(\psi_1, \ldots, \psi_k, \sigma)\downarrow$ and $\mathbf{F}(\varphi_1, \ldots, \varphi_k, \sigma) \simeq \mathbf{F}(\psi_1, \ldots, \psi_k, \sigma)$.

4.1.7 Definition. A functional $\mathbf{F}$ is *partial recursive in a list* $\mathbf{L}$ if there is an index e such that for all $\varphi_1, \ldots, \varphi_k, \sigma$

$$\mathbf{F}(\varphi_1, \ldots, \varphi_k, \sigma) \simeq \{e\}_{\mathbf{L}, \varphi_1 \ldots \varphi_k}(\sigma).$$

$\mathbf{F}$ is *weakly partial recursive* in $\mathbf{L}$ if there is a primitive recursive function $f(n_1, \ldots, n_k)$ such that for all $e_1, \ldots, e_k, \sigma_1, \ldots, \sigma_k$, where $\mathrm{lh}(\sigma_i) = n_i$, $i = 1, \ldots, k$,

$$\mathbf{F}(\lambda\tau_1 \cdot \{e_1\}_{\mathbf{L}}(\tau_1, \sigma_1), \ldots, \lambda\tau_k \cdot \{e_k\}_{\mathbf{L}}(\tau_k, \sigma_n), \sigma) \simeq \{f(n_1, \ldots, n_k)\}_{\mathbf{L}}(e_1, \ldots, e_k, \sigma_1, \ldots, \sigma_k, \sigma).$$

(The reader should compare this with the distinction between weak and strong computability in Section 1.1.7.)

4.1.8 First Recursion Theorem. *Let* $\mathbf{F}$ *be monotone and weakly recursive in* $\mathbf{L}$. *Then there is a least* φ *such that for all* σ, $F(\varphi, \sigma) \simeq \varphi(\sigma)$, *and this* φ *is partial recursive in* $\mathbf{L}$.

The proof is standard and therefore omitted, see the similar proofs in 1.7 and 2.3.

4.1.9 Second Recursion Theorem. *For all indices* e *there is some* x *such that for all lists* $\mathbf{L}$ *and all* σ

$$\{e\}_{\mathbf{L}}(x, \sigma) \simeq \{x\}_{\mathbf{L}}(\sigma).$$

The proof is again standard, see 1.2.6.

We will also need a fixed-point theorem for the class PRF, which is generated by clauses I–XII of 4.1.4. Let $\{e\}_{\mathbf{PRF}}$ be the primitive recursive function with index e. We have the following result.

4.1.10 PRF **Recursion Theorem.** *Let* $f(e, \sigma)$ *be* PRF. *Then there is an index* e *such that for all* σ

$$f(e, \sigma) = \{e\}_{\mathrm{PRF}}(\sigma).$$

It remains to verify that every **F** which is partial recursive in **L** also is weakly partial recursive in **L**. This is an immediate consequence of the following lemma. Note that the converse is not true, as a simple cardinality argument shows.

4.1.11 Lemma. *Let* $\varphi = \lambda\tau\cdot\{e\}_{\mathbf{L}}(\tau, \sigma)$. *There is a primitive recursive function* f *such that for all* x, σ', y:

$$\{x\}_{\mathbf{L},\varphi}(\sigma') \simeq y \quad \textit{iff} \quad \{f(x)\}_{\mathbf{L}}(\sigma', \sigma) \simeq y.$$

We indicate briefly the proof: We define a primitive recursive function g by cases, one for each clause in the inductive definition of $\{x\}_{\mathbf{L},\varphi}(\sigma')$. Let $\mathrm{lh}(\tau) = k$, $\mathrm{lh}(\sigma) = l$, $\mathrm{lh}(\sigma') = n$.

I. $x = \langle 1, n + 1\rangle$: Let $g(x, t) = \langle 1, n + l + 1\rangle$.

Clauses II–IX are treated similarly.

X. $x = \langle 10, n, e, e'\rangle$: Let $g(x, t) = \langle 10, n + l, g(e, t), g(e', t)\rangle$.

We omit XI and XII.

XIII. $x = \langle 13, n + 1\rangle$: There is a primitive recursive function h such that for all t, r, σ'', **L**: $\{h(t)\}_{\mathbf{L}}(r, \sigma'') \simeq \{\{t\}_{\mathrm{PRF}}(r)\}_{\mathbf{L}}(\sigma'')$. Let $g(x, t) = h(t)$.

XV. (Introduction of φ.) $x = \langle 15, k + n, i\rangle$. There is a primitive recursive function s such that for all σ'' of length n: $\{e\}_{\mathbf{L}}(\tau, \sigma) \simeq \{s(e, n)\}_{\mathbf{L}}(\tau, \sigma'', \sigma)$. Let $g(x, t) = s(e, n)$, where $n + k = \mathrm{lh}(\tau, \sigma'')$.

We now use the fixed-point Theorem 4.1.10 for PRF to get a t_0 such that $g(x, t_0) = \{t_0\}_{\mathrm{PRF}}(x)$ for all x. We define $f(x) = g(x, t_0)$, and prove the equivalence of the lemma by inductions on $|\{x\}_{\mathbf{L},\varphi}(\sigma')|_{\mathbf{L},\varphi}$ and $|\{f(x)\}_{\mathbf{L}}(\sigma', \sigma)|_{\mathbf{L}}$, respectively.

4.2 Recursion in a Normal List

In the first section of this chapter we have constructed a computation theory PR(**L**) on domains of the type $\mathfrak{A} = \langle A, S, \mathbf{S}\rangle$. In this section we shall study computation theories which correspond to Kleene recursion in normal objects of higher types. In Chapter 3 we studied the abstract version of recursion in normal type-2 objects. The prototype for the following theory is Kleene recursion in ${}^3E$ on the integers and the reals.

4.2.1 Definition. A list **L** is called *normal* if the functional $\mathbf{E}'_A$ is weakly recursive in **L**, the equality relation on S is recursive in **L**, and **L** contains no partial functions. Remember that $\mathbf{E}'_A$ is defined on total functions $f: A \to S$.

4.2.2 Remarks. 1. Usually "normal" is defined in a stronger sense, viz. by using "recursive in **L**" in place of "*weakly* recursive in **L**". The proofs show that the weak notion suffices in most cases; we shall note an exception below.

2. In the concrete setting of higher types one need *not* require that the equality relation on S, which in this case is $\mathrm{Tp}(0) \cup \ldots \cup \mathrm{Tp}(n-1)$, is recursive in **L**.

3. We remind the reader that if **L** is normal, then the relations recursive in **L** are closed under $\forall$ and $\exists$.

For normal lists **L** we have the basic prewellordering theorem and the selection theorem as for Spector theories, see Chapter 3. The proofs are the same, hence we only state the relevant results.

4.2.3 Theorem. *Let* PR(**L**) *be a normal theory on* $\mathfrak{A} = \langle A, S, \mathbf{S}\rangle$. *Then* PR(**L**) *is p-normal, i.e. there is a function p partial recursive in* PR(**L**) *such that*

(i) $x \in \mathbf{C}_\mathbf{L}$ *or* $y \in \mathbf{C}_\mathbf{L}$ *iff* $p(x, y)\downarrow$,
(ii) $x \in \mathbf{C}_\mathbf{L}$ *and* $|x|_\mathbf{L} \leqslant |y|_\mathbf{L} \Rightarrow p(x, y) = 0$,
$|x|_\mathbf{L} > |y|_\mathbf{L} \Rightarrow p(x, y) = 1$.

Here $\mathbf{C}_\mathbf{L} = \{\langle e, \sigma\rangle : \{e\}_\mathbf{L}(\sigma)\downarrow\}$ is the set of computations in PR(**L**).

4.2.4 Theorem. *Let* PR(**L**) *be normal on* $\mathfrak{A} = \langle A, S, \mathbf{S}\rangle$. *There is a function* $\varphi \in$ PR(**L**) *such that for all* e, σ: *If*

$$\exists n \in N\{e\}_\mathbf{L}(n, \sigma)\downarrow \Rightarrow \varphi(e, \sigma)\downarrow \quad \text{and} \quad \{e\}_\mathbf{L}(\varphi(e, \sigma), \sigma)\downarrow .$$

Moreover if $\varphi(e, \sigma) \simeq n$, *then* $\{e\}_\mathbf{L}(n, \sigma)\downarrow$. *The index of* φ *is primitive recursive in* $\mathrm{lh}(\sigma)$.

4.2.4 is obtained by the standard argument from 4.2.3, see the proof of 3.1.6. We have the usual corollaries, i.e. the PR(**L**)-semicomputable relations are closed under $\vee$ and $\exists n \in N$.

We shall now prove that the PR(**L**)-semicomputable relations are *not* closed under $\exists x \in \mathrm{Tp}(S)$. An important result about normal theories PR(**L**) is that they are closed under $\exists s \in S$; this will be the topic of the next section.

We first need more precise information about subcomputations and computation trees.

4.2.5 Lemma. *Let* $=_S$ *be recursive in* **L** *and* $\mathbf{E}'_S$ *weakly recursive in* **L**. *There is a relation* $S(x, y)$ *semicomputable in* PR(**L**), *such that if* $x \in \mathbf{C}_\mathbf{L}$ *then*

$$S(x, y) \quad \textit{iff} \quad y \textit{ is an immediate subcomputation of } x.$$

The set $\{y : S(x, y)\}$ *is recursive in* x, **L** *when* $x \in \mathbf{C}_\mathbf{L}$.

The proof of the lemma comes from an analysis of the clauses of the inductive definition of PR(**L**); see Definition 4.1.4. The assumptions on **L** are required so as to decide the form of x, i.e. to decide the form of the index of the tuple. To do this we need e.g. the predicate Seq., see Lemma 4.1.3. Otherwise the proof is routine.

4.2.6 Lemma. *Let* **L** *be normal. If* $\{e\}_{\mathbf{L}}(\sigma)\downarrow$, *then the computation tree of* $\langle e, \sigma\rangle$ *is recursive in* **L**, σ.

Proof. Let q be the least fixed-point for the monotone **L**-recursive functional **F** defined as follows:

$$\begin{aligned}\mathbf{F}(q, x, y) \simeq 0 \quad &\text{if} \quad x \in \mathbf{C}_{\mathbf{L}} \quad \text{and} \quad (S(x, y) \vee \exists z[S(x, z) \\ &\qquad\qquad \wedge q(z, y) \simeq 0]) \\ \simeq 1 \quad &\text{if} \quad x \in \mathbf{C}_{\mathbf{L}} \quad \text{and} \quad (\neg S(x, y) \wedge \forall z[S(x, z) \\ &\qquad\qquad \rightarrow q(z, y) \simeq 1]).\end{aligned}$$

Note that the quantifiers in **F** can be handled by $\mathbf{E}'_A$. We now appeal to the first recursion theorem and obtain a solution q such that $q(x, y)\downarrow$ iff $x \in \mathbf{C}_{\mathbf{L}}$.

If $x \in \mathbf{C}_{\mathbf{L}}$, then $\lambda y \cdot q(\langle e, \sigma\rangle, y)$ is the (**L**-recursive) characteristic function of the computation tree of $\{e\}_{\mathbf{L}}(\sigma)$.

In connection with the coding functions on $\mathfrak{A}$ we introduced two mappings $^*: S \to \mathrm{Tp}(S)$ and $^-: A \to S$. We shall need some further coding and translation devices.

First let Y be a set of elements in $\mathrm{Tp}(S)$ indexed by S, i.e. $Y = \{\alpha_r : r \in S\}$. Then all elements in Y can be coded by *one* element in $\mathrm{Tp}(S)$,

$$\alpha(r) = \alpha_{(r)_1}((r)_2).$$

For all $r \in S$, $\lambda s \cdot \alpha(\langle r, s\rangle) = \alpha_r$.

Further there is a one-one mapping $^{**}: A \to \mathrm{Tp}(S)$ and a mapping $^{--}: A \to A$ such that the graphs of ** and $^{--}$ are primitive recursive in $=_S$ and $\mathbf{E}'_S$ and such that $(x^{**})^{--} = x$, for all $x \in A$. One way of defining ** and $^{--}$ is as follows:

$$\begin{aligned}r^{**} &= \langle r^*, \mathbf{0}\rangle \\ \alpha^{**} &= \langle \alpha, \mathbf{1}\rangle,\end{aligned}$$

where $\mathbf{0}, \mathbf{1} \in \mathrm{Tp}(S)$ are the constant functions. And

$$x^{--} = \begin{cases}(x)_1 & \text{if} \quad (x)_2(0) = 1 \\ (x)_1^- & \text{if} \quad (x)_2(0) = 0.\end{cases}$$

4.2.7 Theorem. *Let* $=_S$ *be* **L**-*recursive and* $\mathbf{E}'_S$ *weakly* **L**-*recursive. There is* PR(**L**)-*semicomputable relation* R *such that for all* e, σ:

$$\{e\}_{\mathbf{L}}(\sigma)\uparrow \quad \textit{iff} \quad \exists\alpha R(\alpha, \langle e, \sigma\rangle).$$

Proof: We use the fact of Theorem 4.1.6: $\{e\}_{\mathbf{L}}(\sigma)$ diverges iff the computation tree of $\langle e, \sigma\rangle$ is not well-founded. The condition for this is that we have an infinite descending chain of immediate subcomputations of $\langle e, \sigma\rangle$, i.e.

$$\exists\alpha_0\exists\alpha_1 \ldots \exists\alpha_n \ldots [\alpha_0^{--} = \langle e, \sigma\rangle \wedge \forall i \cdot S(\alpha_i^{--}, \alpha_{i+1}^{--})].$$

By the remark above the chain can be coded by one element of $\mathrm{Tp}(S)$. The sought for relation R is then simply

$$R(\alpha, \langle e, \sigma\rangle) \quad \text{iff} \quad \alpha[0]^{--} = \langle e, \sigma\rangle \wedge \forall i \cdot S(\alpha[i]^{--}, \alpha[i+1]^{--}),$$

where S comes from Lemma 4.2.5 and $\alpha[r] = \lambda s \cdot \alpha(\langle r, s\rangle)$.

4.2.8 Corollary. *The relations which are* PR(L)*-semicomputable are not closed under* $\exists\alpha \in \mathrm{Tp}(S)$ *when* **L** *is a normal list.*

4.2.9 Remark. So far we have needed only weak L-recursiveness. Here is one example where (strong) L-recursiveness is necessary. Call a relation $R \subseteq 2^A \times A$ recursive in **L** if there is an index e such that $\lambda X\lambda x \cdot \{e\}_{\mathbf{L},X}$ is the characteristic function of R.

We have the following normal form theorem: Let **L** be normal and assume that $\mathbf{E}'_A$ is recursive in **L**. Let $B \subseteq A$: Then B is **L**-semicomputable iff there exists $R \subseteq 2^A \times A$ which is recursive in **L** and such that for all $x \in A$:

(*) $\quad x \in B$ iff $\exists X(X$ is recursive in $x, \mathbf{L} \wedge R(X, x))$.

Note that this is reminiscent of the Gandy-Spector theorem, but far weaker, however. In the Gandy-Spector theorem the relation R is arithmetic over the domain.

The proof is omitted, but can be found in Moldestad [105], page 40. We note that the X on the RHS of (*) should be the computation tree of the computation $\{e_0\}_{\mathbf{L}}(x) \simeq 0$, which asserts that $x \in B$. This tree is recursive by Lemma 4.2.6. And we need the strong recursiveness of $\mathbf{E}'_A$ since in the analysis of the tree we meet conditions of the form $\forall x(\ldots x, X, \ldots)$. Weak recursiveness cannot handle such clauses.

4.3 Selection in Higher Types

We now come to one of the most important results about PR(L), where **L** is a normal list on a domain of two types $\mathfrak{A} = \langle A, S, \mathbf{S}\rangle$. As we remarked in 4.2.4,

PR(**L**) admits selection operators over N, hence the PR(**L**)-semicomputable relations are closed under $\exists n \in N$. In 4.2.8 we saw that the class is not closed under $\exists \alpha \in \mathrm{Tp}(S)$. In this section we prove that the class is closed under $\exists s \in S$.

4.3.1 Theorem. *Let* **L** *be a normal list on* $\mathfrak{A} = \langle A, S, \mathbf{S}\rangle$. *There is a function* φ *partial recursive in* **L** *with index* $\hat{\mathrm{e}}$ *such that if*

$$\exists x \in S \cdot \{e\}_{\mathbf{L}}(x, \sigma)\downarrow,$$

then $\varphi(\langle e, \sigma\rangle)\downarrow$, *and in this case*

$$|\{\hat{\mathrm{e}}\}_{\mathbf{L}}(\langle e, \sigma\rangle)|_{\mathbf{L}} \geqslant \min_{x \in S} \{|\{e\}_{\mathbf{L}}(x, \sigma)|_{\mathbf{L}}\}.$$

Conversely, if $\varphi(\langle e, \sigma\rangle)\downarrow$, *then* $\exists x \in S \cdot \{e\}_{\mathbf{L}}(x, \sigma)\downarrow$.

This theorem immediately entails the closure of the PR(**L**)-semirecursive relations under $\exists s \in S$. As we have remarked before, the existence of a single-valued selection operator is closely connected to the existence of a computable well-ordering of the domain. The next best thing without going all the way to multiple-valued theories, is to be able to compute something, in 4.3.1 $\varphi(\langle e, \sigma\rangle)$, which effectively gives us a non-empty computable subset of a given semicomputable set. (The reader should refer back to our discussion in connection with 3.1.10.)

We spell out the details: Let B be an **L**-semicomputable subset of S, i.e.

$$B = \{x \in S : \{e\}_{\mathbf{L}}(x, \sigma)\downarrow\}.$$

We note that $\varphi(\langle e, \sigma\rangle)\downarrow$ iff $B \neq \varnothing$. And if $B \neq \varnothing$, then

$$B_0 = \{x \in S : |\{e\}_{\mathbf{L}}(x, \sigma)|_{\mathbf{L}} \leqslant |\varphi(\langle e, \sigma\rangle)_{\mathbf{L}}|\},$$

is a non-empty **L**-computable subset of B. Obviously, an index for B_0 can be computed in PR(**L**) from the given data.

The selection principle involved in 4.3.1 was first stated by Grilliot [48] in the context of a theory of inductive definability. His proof was, however, defective. A first proof in the context of Kleene recursion in higher types was given in the thesis of D. MacQueen [98], and was published in a somewhat abstract version in L. Harrington and D. MacQueen [55]. A proof, based on MacQueen's thesis, was worked out by Moldestad [105] in the framework of computation theories on two types. It is this proof that we will present in this section.

The remainder of this section is devoted to a proof of Theorem 4.3.1. We make one simple reduction. The set $\{\langle e, x, \sigma\rangle^{**} : x \in S\}$ is a family of elements of $\mathrm{Tp}(S)$ indexed by S, and hence can be coded by one element $\alpha \in \mathrm{Tp}(S)$. Then

$$\exists s \in S \cdot \{e\}_{\mathbf{L}}(s, \sigma)\downarrow \quad \text{iff} \quad \exists s \in S \cdot \alpha[s]^{--} \in \mathbf{C}_{\mathbf{L}},$$

where $\mathbf{C}_{\mathbf{L}}$ is the set of convergent computations, and as above $\alpha[s] = \lambda r \cdot \alpha(\langle s, r\rangle)$.

4.3.2 Definition. For any $\beta \in \mathrm{Tp}(S)$ let $\|\beta\| = \min\{|\beta[s]^{--}|_{\mathbf{L}} : s \in S\}$.

We see that in order to prove Theorem 4.3.1 it suffices to prove the following lemma.

4.3.3 Lemma. *There is an index m such that*

(i) $\|\beta\| < \kappa_{\mathbf{L}} \Rightarrow \{m\}_{\mathbf{L}}(\beta)\downarrow \wedge \|\beta\| < |\{m\}_{\mathbf{L}}(\beta)|$.
(ii) $\{m\}_{\mathbf{L}}(\beta)\downarrow \Rightarrow \|\beta\| < \kappa_{\mathbf{L}}$.

The proof will use the recursion theorem. We shall, by induction on the norm of β, i.e. on $\|\beta\|$, show how to define $\{m\}_{\mathbf{L}}(\beta)$. We assume as induction hypothesis

$$\|\beta\| < \mu \Rightarrow \{m\}_{\mathbf{L}}(\beta)\downarrow \wedge \|\beta\| \leqslant |\{m\}_{\mathbf{L}}(\beta)|_{\mathbf{L}}.$$

Assume that $\alpha \in \mathrm{Tp}(S)$ is such that $\|\alpha\| = \mu$. We shall show how to define $\{m\}_{\mathbf{L}}(\alpha)$. Note that α is kept fixed for the rest of the proof.

Intuitively, α is meant to code a family of computations $\{\langle e, x, \sigma\rangle : x \in S\}$. In order to compute something which dominates some computation in this family, we need to go into a detailed analysis of subcomputations. Lemma 4.2.5 tells us that there is an **L**-semicomputable relation $S(x, y)$ such that if $x \in \mathbf{C}_{\mathbf{L}}$, then y is an immediate subcomputation of x iff $S(x, y)$.

Except for *substitution*, the notion of *immediate subcomputation* is unproblematic. In the following analysis we control the search for immediate subcomputations through the following "truncated" version of $S(x, y)$: Let $R(x, y, w)$ be a relation such that if $w \in \mathbf{C}_{\mathbf{L}}$:

1 If x is not a substitution, i.e. x is not of the form $\langle\langle 10, n, e, e'\rangle, \sigma\rangle$, then $\{y : R(x, y, w)\} = \{y : S(x, y)\}$.
2 If x is a substitution, i.e. $x = \langle\langle 10, n, e, e'\rangle, \sigma\rangle$, then

$$\{y : R(x, y, w)\} = \begin{cases} \{\langle e, \sigma\rangle\} & \text{if } |w|_{\mathbf{L}} < |\{e\}_{\mathbf{L}}(\sigma)|_{\mathbf{L}}, \\ \{\langle e, \sigma\rangle, \langle e', \{e\}_{\mathbf{L}}(\sigma), \sigma\rangle\} & \text{otherwise.} \end{cases}$$

We see how we use a $w \in \mathbf{C}_{\mathbf{L}}$ as a "cut-off" measure in searching for the immediate subcomputations of x. Note that if $x \in \mathbf{C}_{\mathbf{L}}$, then there is some $w \in \mathbf{C}_{\mathbf{L}}$ such that $R(x, y, w)$ iff $S(x, y)$.

For $\sigma < \kappa_{\mathbf{L}}$ we introduce a set

$$T_\sigma = \{\beta : \forall x \in S \cdot R(\alpha[x]^{--}, \beta[x]^{--}, w)\},$$

where $|w|_{\mathbf{L}} = \sigma$.

Note. At this point there is a contradiction in our notation. But don't make a fuss! Even if our reader refuses to go into the details of the proof, he or she should have no difficulties in correctly classifying the occurrences of the letter σ, whether it is an ordinal or an input sequence of a computation.

Any $\beta \in T_\sigma$ is intended to represent a family of computations which up to the order of approximation $\sigma = |w|_{\mathbf{L}}$ are "pointwise" immediate subcomputations of the family α. Note that

$$\beta \in T_\sigma \Rightarrow \|\beta\| < \|\alpha\|$$
$$\sigma < \tau \Rightarrow \mathbf{T}_\sigma \subseteq \mathbf{T}_\tau.$$

We observe that T_σ is recursive in $\mathbf{L}$, α, w where $|w|_{\mathbf{L}} = \sigma$.

It is clear that by searching through "enough" ordinals σ we can decide whether any β codes a family of immediate subcomputations of α. What "enough" means is made precise in the following lemma.

4.3.4 Lemma. *Let λ be an ordinal such that S is not cofinal in λ. Let $\{\sigma(\tau) : \tau < \lambda\}$ be an increasing sequence of ordinals bounded by $\kappa_{\mathbf{L}}$. Then there exists an ordinal $\tau' < \lambda$ such that*

$$T_{\sigma(\tau)} = T_{\sigma(\tau')},$$

for all τ satisfying $\tau' \leqslant \tau < \lambda$.

We postpone the proof of the lemma. We shall apply it in the following situation. Let

$$W = \{\gamma : \mathrm{PWO}(\gamma) \wedge \mathrm{dom}(\gamma) \subseteq S\}.$$

W is recursive in $\mathbf{L}$—we have prewellorderings of S, not of $A = S \cup \mathrm{Tp}(S)$. This is the point where an attempt to extend the result from $\exists x \in S$ to $\exists x \in A$ would fail, and necessarily so as 4.2.8 shows.

For $\delta \in W$, let $\mathrm{or}(\delta)$ = length of the pwo δ. Letting $\lambda = \sup\{\mathrm{or}(\delta) : \delta \in W\}$, we observe that S is not cofinal in λ.

We are now ready for the inductive computation of $\{m\}_{\mathbf{L}}(\alpha)$:

(i) There exists an index m_1 such

$$\{m_1\}_{\mathbf{L}}(m, \alpha, w)\downarrow \quad \text{iff} \quad w \in \mathbf{C}_{\mathbf{L}} \wedge \{m\}_{\mathbf{L}}(\beta)\downarrow, \quad \text{for all } \beta \in T_{|w|}$$

and if $w \in \mathbf{C}_{\mathbf{L}}$, then

$$|\{m_1\}_{\mathbf{L}}(m, \alpha, w)|_{\mathbf{L}} > |\{m\}_{\mathbf{L}}(\beta)|_{\mathbf{L}}, \quad \text{for all } \beta \in T_{|w|}.$$

We note that it follows from the induction hypothesis that $|\{m_1\}_{\mathbf{L}}(m, \alpha, w)| > \|\beta\|$ for all $\beta \in T_{|w|}$.

(ii) There is an index m_2 (using the recursion theorem) such that

$$\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma)\downarrow \quad \text{iff} \quad \gamma \in W \wedge \forall \gamma' \in W[\mathrm{or}(\gamma') < \mathrm{or}(\gamma) \Rightarrow \{m_2\}_{\mathbf{L}}(m, \alpha, \gamma')\downarrow \wedge \{m_1\}_{\mathbf{L}}(m, \alpha, \langle m_2, m, \alpha, \gamma'\rangle)\downarrow].$$

m_2 can be chosen such that for all $\gamma' \in W$ with $\mathrm{or}(\gamma') < \mathrm{or}(\gamma)$:

$$|\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma)|_{\mathbf{L}} > |\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma')|_{\mathbf{L}}, |\{m_1\}_{\mathbf{L}}(m, \alpha, \langle m_2, m, \alpha, \gamma' \rangle)|_{\mathbf{L}}.$$

It follows from the induction hypothesis that $|\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma)| > \|\beta\|$ for all $\beta \in T_{|\langle m_2, m, \alpha, \gamma' \rangle|}$, provided $\mathrm{or}(\gamma') < \mathrm{or}(\gamma)$.

(iii) There is an index m_3 such that

$$\{m_3\}_{\mathbf{L}}(m, \alpha)\downarrow \quad \text{iff} \quad \forall \gamma \in W \{m_2\}_{\mathbf{L}}(m, \alpha, \gamma)\downarrow,$$

and such that whenever $\gamma \in W$

$$|\{m_3\}_{\mathbf{L}}(m, \alpha)| > |\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma)|.$$

By the recursion theorem we now find an index m such that

$$\{m\}_{\mathbf{L}}(\alpha) \simeq \{m_3\}_{\mathbf{L}}(m, \alpha),$$

and

$$|\{m\}_{\mathbf{L}}(\alpha)| \geqslant |\{m_3\}_{\mathbf{L}}(m, \alpha)|.$$

Part (i) of Lemma 4.3.3 now follows if we can show that $|\{m_3\}_{\mathbf{L}}(m, \alpha)| \geqslant \|\alpha\|$.

This will follow from Lemma 4.3.4 applied to the ordinal λ derived from the set of prewellorderings W. In detail, define for each $\tau < \lambda$

$$\sigma(\tau) = \inf\{|\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma)|_{\mathbf{L}}; \gamma \in W \wedge \mathrm{or}(\gamma) = \tau\}.$$

Then $\{\sigma(\tau) : \tau < \lambda\}$ is an increasing sequence bounded by $|\{m_3\}_{\mathbf{L}}(m, \alpha)|_{\mathbf{L}} < \kappa_{\mathbf{L}}$. Lemma 4.3.4 applies, i.e. there is an ordinal $\tau' < \lambda$ such that $T_{\sigma(\tau)} = T_{\sigma(\tau')}$ whenever $\tau' \leqslant \tau < \lambda$. Let $\sigma = \sup\{\sigma(\tau); \tau < \lambda\}$. The induction hypothesis gives $|\{m_3\}_{\mathbf{L}}(m, \alpha)|_{\mathbf{L}} \geqslant \sigma$. The ordinal of a computation is determined in terms of the ordinals of the immediate subcomputations. We have controlled the immediate subcomputations of the family α by the sets $T_{\sigma(\tau)}$ and the ordinal σ, hence by the computation $|\{m_3\}_{\mathbf{L}}(m, \alpha)|_{\mathbf{L}}$. So, we expect, and it only formally remains to verify:

4.3.5 Claim. $\sigma \geqslant \|\alpha\|$,

to end the proof of Lemma 4.3.3 (i). Part (ii) of the lemma will then be proved by induction on the length $|\{m\}_{\mathbf{L}}(\alpha)|$.

The reader who is not particularly interested in the fine details of this "cleaning-up-business" may proceed to the next section.

4.3.6 Proof of Lemma 4.3.4. We proceed by contradiction, i.e. assume that $\forall \tau' < \lambda$ $\exists \tau(\tau' \leqslant \tau < \lambda \wedge T_{\sigma(\tau')} \subsetneq T_{\sigma(\tau)})$, and construct a function $f: S \to \lambda$ such that $\lambda = \sup\{f(x) : x \in S\}$.

Take $\tau' < \lambda$ and choose τ_0 minimal such that $\tau' \leqslant \tau_0$ and $T_{\sigma(\tau')} \subsetneqq T_{\sigma(\tau_0)}$. Obviously $\tau' < \tau_0$. Choose $w, w' \in \mathbf{C}_{\mathbf{L}}$ such that $\tau' = |w'|$ and $\tau_0 = |w|$.

If $\beta \in T_{\sigma(\tau_0)} - T_{\sigma(\tau')}$, then

$$\forall x \in S \cdot R(\alpha[x]^{--}, \beta[x]^{--}, w)$$
$$\exists x \in S \cdot \neg R(\alpha[x]^{--}, \beta[x]^{--}, w').$$

If $\neg R(\alpha[x]^{--}, \beta[x]^{--}, w')$ but $R(\alpha[x]^{--}, \beta[x]^{--}, w)$, then $\alpha[x]^{--}$ is a substitution $\langle\langle 10, n, e, e'\rangle, \sigma\rangle$, $\beta[x]^{--} = \langle e', \{e\}_{\mathbf{L}}(\sigma), \sigma\rangle$, and $|w'| < |\{e\}_{\mathbf{L}}(\sigma)| \leqslant |w|$. Hence $R(\alpha[x]^{--}, \beta[x]^{--}, w'')$ for all $|w''| \geqslant |w|$.

Let

$$P(\tau') = \{x \in S : \exists \beta \in T_{\sigma(\tau_0)} - T_{\sigma(\tau')}, \neg R(\alpha[x]^{--}, \beta[x]^{--}, w')\}.$$

From the remarks above it follows that $P(\tau') \neq \varnothing$, but $P(\tau') = P(\nu)$, $\tau' \leqslant \nu < \tau_0$; and that $P(\tau') \cap P(\nu) = \varnothing$ if $\nu \geqslant \tau_0$.

Let $f\colon S \to A$ be defined by

$$f(x) = \begin{cases} \text{least } \tau' \text{ such that } x \in P(\tau'), & \text{if } \ x \in \bigcup P(\tau),\ \tau < \lambda \\ 0, & \text{otherwise.} \end{cases}$$

Clearly, $\sup\{f(x) : x \in S\} = \lambda$.

4.3.7 Proof of Claim 4.3.5. We argue, once more, by contradiction, i.e. we assume that $\sigma < \|\alpha\|$. We keep the notations from the claim: in particular, the ordinals τ' and σ have their established meaning.

We make a preliminary remark: Let $x \in S$. If $\alpha[x]^{--}$ is a substitution, then either $|\{e\}_{\mathbf{L}}(\sigma)|_{\mathbf{L}} \leqslant \sigma(\tau')$ or $\sigma \leqslant |\{e\}_{\mathbf{L}}(\sigma)|_{\mathbf{L}}$. (For if $\sigma(\tau') < |\{e\}_{\mathbf{L}}(\sigma)|_{\mathbf{L}} < \sigma$, take a $\beta \in T_{\sigma(\tau')}$ and let $\beta'[y] = \beta[y]$ if $y \neq x$, $\beta'[x]^{--} = \langle e', \{e\}_{\mathbf{L}}(\sigma), \sigma\rangle$. Then $\beta' \in T_{\sigma(\tau')}$. But $\beta' \in T_{\sigma(\tau)}$ for $\sigma(\tau) > |\{e\}_{\mathbf{L}}(\sigma)|_{\mathbf{L}}$. This contradicts the fact that $T_{\sigma(\tau)} = T_{\sigma(\tau')}$.)

Using the assumption $\sigma < \|\alpha\|$ we construct a β in the following way:

(i) If $\alpha[x]^{--}$ is not a substitution, let $\beta[x]^{--}$ be such that $S(\alpha[x]^{--}, \beta[x]^{--})$ and such that $|\beta[x]^{--}| \geqslant \sigma$.

The inequality uses the fact that $\sigma < \|\alpha\| = \min\{|\alpha[x]^{--}|_{\mathbf{L}} : x \in S\}$.

(ii) If $\alpha[x]^{--}$ is a substitution $\langle\langle 10, n, e, e'\rangle, \sigma\rangle$ let

$$\beta[x]^{--} = \begin{cases} \langle e', \{e\}_{\mathbf{L}}(\sigma), \sigma\rangle & \text{if } \ |\{e\}_{\mathbf{L}}(\sigma)| \leqslant \sigma(\tau') \\ \langle e, \sigma\rangle & \text{if } \ |\{e\}_{\mathbf{L}}(\sigma)| \geqslant \sigma \end{cases}$$

By construction $\beta \in T_{\sigma(\tau')}$.

We now claim that $\|\beta\| \geqslant \sigma$: By definition $\|\beta\| = \min\{|\beta[x]|^{--} : x \in S\}$. From (i) and (ii) we see that $|\beta[x]^{--}| \geqslant \sigma$ in all cases except possibly when $|\{e\}_{\mathbf{L}}(\sigma)|_{\mathbf{L}} \leqslant \sigma(\tau')$. But even in this case $|\beta[x]^{--}| = |\{e'\}_{\mathbf{L}}(\{e\}_{\mathbf{L}}(\sigma), \sigma)| \geqslant \sigma$, because otherwise $|\alpha[x]^{--}| \leqslant \sigma$, which contradicts the assumption $\sigma < \|\alpha\|$.

Combining this claim with the fact that $\beta \in T_{\sigma(\tau')}$ by construction, we derive the final contradiction: Choose τ such that $\tau' < \tau < \lambda$. Choose $\gamma', \gamma \in W$ such that $\mathrm{or}(\gamma') = \tau'$, $\mathrm{or}(\gamma) = \tau$, $\sigma(\tau') = |\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma')|$, and $\sigma(\tau) = |\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma)|$.

By construction of m_2, $|\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma) \geqslant \|\beta'\|$ for all $\beta' \in T_{|\langle m_2, m, \alpha, \gamma' \rangle|}$. In particular, $|\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma)| \geqslant \|\beta\|$, since $\beta \in T_{\sigma(\tau')} = T_{|\langle m_2, m, \alpha, \gamma' \rangle|}$, contradicting the fact that $|\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma)| = \sigma(\tau) < \sigma \leqslant \|\beta\|$.

Note how this proof bears out our remarks in connection with 4.3.5. We have controlled the subcomputations of α by the ordinal σ, hence $\|\alpha\| \leqslant \sigma$. (See, in particular, the verification that $\|\beta\| > \sigma$.)

4.3.8 Proof of (ii) in Lemma 4.3.3. This is a standard inductive argument on the ordinal of $\{m\}_{\mathbf{L}}(\sigma)$.

Let $\{m\}_{\mathbf{L}}(\alpha)\downarrow$ and assume that 4.3.3 (ii) is satisfied for all β such that $|\{m\}_{\mathbf{L}}(\beta)| < |\{m\}_{\mathbf{L}}(\alpha)|$. Since $\{m\}_{\mathbf{L}}(\alpha)\downarrow$, we have $\{m_3\}_{\mathbf{L}}(m, \alpha)\downarrow$ and $|\{m\}_{\mathbf{L}}(\alpha)| > |\{m_3\}_{\mathbf{L}}(m, \alpha)|$. Further $\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma)\downarrow$ for all $\gamma \in W$ and $|\{m_3\}(m, \alpha)| > |\{m_2\}(m, \alpha, \gamma)|$ for all such γ. Let the ordinals $\{\sigma(\tau) : \tau < \lambda\}$ be defined as above, and choose $\tau' < \lambda$ as before.

We recall that if $\alpha[x]^{--}$ is a substitution, then either $|\{e\}_{\mathbf{L}}(\sigma)| \leqslant \sigma(\tau')$ or $\sigma \leqslant |\{e\}_{\mathbf{L}}(\sigma)|$; see the proof of 4.3.5.

We argue once more by contradiction and so assume that $\|\alpha\| = \kappa_{\mathbf{L}}$, i.e. $\alpha[x]^{--}$ codes a divergent computation for all $x \in S$.

Construct a β as follows: If $\alpha[x]^{--}$ is not a substitution, let $\beta[x]^{--}$ be a divergent subcomputation of $\alpha[x]^{--}$. If $\alpha[x]^{--}$ is a substitution let $\beta[x]^{--}$ be constructed as in (ii) of 4.3.7. By construction $\beta \in T_{\sigma(\tau')}$, and we see that $|\beta[x]^{--}| \geqslant \sigma$ for all $x \in S$; hence $\|\beta\| \geqslant \sigma$.

Choose $\gamma' \in W$ such that $\sigma(\tau') = |\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma')|$, and pick a $\gamma \in W$ such that $\mathrm{or}(\gamma') < \mathrm{or}(\gamma)$. By construction of m_2:

$$|\{m_2\}_{\mathbf{L}}(m, \alpha, \gamma)| > |\{m_1\}_{\mathbf{L}}(m, \alpha, \langle m_2, m, \alpha, \gamma' \rangle)|.$$

By construction of m_1:

$$|\{m_1\}_{\mathbf{L}}(m_1, \alpha, \langle m_2, m, \alpha, \gamma' \rangle| > |\{m\}_{\mathbf{L}}(\beta)|,$$

since $\beta \in T_{\sigma(\tau')} = T_{|\langle m_2, m, \alpha, \gamma' \rangle|}$. Hence $|\{m\}_{\mathbf{L}}(\alpha)| > |\{m\}_{\mathbf{L}}(\beta)|$.

So by the induction hypothesis $\|\beta\| < \kappa_{\mathbf{L}}$. By part (i) of Lemma 4.3.3 it follows that $\|\beta\| \leqslant |\{m\}_{\mathbf{L}}(\beta)|$. Hence

$$\|\beta\| < |\{m_2\}(m, \alpha, \gamma)|,$$

for all $\gamma \in W$ such that $\mathrm{or}(\gamma') < \mathrm{or}(\gamma)$. By definition of $\sigma(\tau)$, $\|\beta\| < \sigma(\tau)$ when $\tau' < \tau < \lambda$. Hence $\|\beta\| < \sigma$, which contradicts the fact that $\|\beta\| \geqslant \sigma$ by construction.

4.4 Computation Theories and Second Order Definability

We shall make a few brief remarks on second-order definability. But first we draw the basic results of Sections 4.1–4.3 together in the following theorem.

4.4.1 Theorem. *Let* PR(L) *be normal on* $A = S \cup \mathrm{Tp}(S)$. *The following is true*

(a) PR(L) *is p-normal, hence admits a selection operator over N,*

(b) *A is weakly but not strongly* L*-finite, i.e. the* L*-semicomputable relations are not closed under* $\exists x \in \mathrm{Tp}(S)$,

(c) *S is strongly* L*-finite, i.e. the* L*-semicomputable relations are closed under* $\exists s \in S$.

As we shall see in Chapter 7, properties (a)–(c) characterize Kleene recursion in a normal object in higher types.

4.4.2 Remark. Let us be a bit more explicit about the relationship between recursion on two types and Kleene recursion in higher types. If $S = \mathrm{Tp}(0) \cup \ldots \cup \mathrm{Tp}(n-1)$, $n > 0$, then $\mathrm{Tp}(S)$ can be identified with $\mathrm{Tp}(1) \times \ldots \times \mathrm{Tp}(n)$. If F is an object of type $n+2$, there is a list L such that Kleene recursion in F on $\mathrm{Tp}(0), \ldots, \mathrm{Tp}(n)$ is essentially the same as recursion in L on the structure $\mathfrak{A} = \langle A, S, \mathbf{S} \rangle$, where S is some standard coding scheme. A converse is also true. Hence results about recursion in higher types can be deduced from the corresponding results for PR(L) on $\mathfrak{A}$. Thus in a quite precise sense, higher type theories can really be captured as theories on two types. (A detailed exposition of the connection between two types and higher types can be found in the book of J. Moldestad [105].)

So far we have emphasized the connection between computation theories on two types and the specific example of recursion in higher types. But the theory has a wider scope and can serve as a framework for the study of second-order definability in general. This is completely analogous to the case of finite theories on one type where recursion in ${}^2E$ was, in a sense, the paradigm, but the theory had much wider connections with definability theory and inductive definability.

An alternate way of approaching second-order definability is via the notion of a *Spector 2-class*. This notion is introduced in Moschovakis [118], a survey of concepts and applications can be found in the lectures of A. Kechris [76]. *There is the same relationship between computation theories on two types and Spector 2-classes as there is between finite theories on one type* (*Spector theories*) *and Spector classes* (see Section 3.2).

We do not give a formal development of this relationship in this book, since the applications of Spector 2-classes all fall under the scope of computation theories on two types. However, we strongly recommend that the reader study the lectures of Kechris [76]. And may we suggest that he or she tries to lift from one to two types the development presented in Sections 3.2 and 3.3, i.e. explore the relationship between computation theories on two types and Spector 2-classes and see how the applications come out in the context of two types. The groundwork is done in Sections 4.1 to 4.3, and we shall present selected applications in Chapters 7 and 8.