NOTIONS OF COMPUTABILITY AT HIGHER TYPES I

JOHN R. LONGLEY

Abstract. This is the first of a series of three articles devoted to the conceptual problem of identifying the natural notions of computability at higher types (over the natural numbers) and establishing the relationships between these notions. In the present paper, we undertake an extended survey of the different strands of research to date on higher type computability, bringing together material from recursion theory, constructive logic and computer science, and emphasizing the historical development of the ideas. The paper thus serves as a reasonably comprehensive survey of the literature on higher type computability.

CONTENTS

1. Introduction 3
1.1. Motivations 6
1.1.1. Constructive logic and metamathematics 6
1.1.2. Descriptive and admissible set theory 6
1.1.3. Abstract computability theories 7
1.1.4. Semantics and logic of programming languages 7
1.1.5. Subrecursion and complexity 8
1.1.6. Real number computability 8
1.1.7. Computability in physics 9
1.2. Overview of the series 9
1.3. Outline of the present paper 10
1.4. Notation 11
1.5. Prerequisites 13
1.5.1. Types and type structures 13
1.5.2. The λ-calculus 14
1.5.3. Cartesian closed categories 16
1.5.4. Historical remarks 17
2. Early work: Computability at type 2 18
2.1. Prehistory 18
2.2. Banach-Mazur functionals 18
2.3. Computations on pure functions 18
2.4. Computations on Kleene indices 21
3. Total computable functionals 24
3.1. System T and related type systems 24

Logic Colloquium '00
Edited by René Cori, Alexander Razborov, Stevo Todorčević, and Carol Wood
Lecture Notes in Logic, 00

3.1.1. Gödel's T 25
3.1.2. Kleene's S1–S8 27
3.2. Kleene computability: S1–S9 28
3.2.1. Kleene's work 28
3.2.2. Recursion in normal objects 33
3.2.3. Hierarchies 36
3.2.4. Set recursion 37
3.3. The total continuous functionals 38
3.3.1. Early work: Kleene and Kreisel 38
3.3.2. Further characterizations 41
3.3.3. Degrees and relative computability 42
3.3.4. Recent work 43
3.3.5. Type two effectivity 44
3.4. The hereditarily effective operations 44
4. Partial computable functionals 47
4.1. Partial monotone functionals 48
4.1.1. Platek's thesis 49
4.1.2. Kleene's later work 52
4.2. Partial continuous functionals 55
4.2.1. Scott's approach 55
4.2.2. Ershov's approach 57
4.2.3. Later developments 59
4.2.4. PCF and parallelism 60
4.2.5. Operational semantics 63
4.3. PCF and sequential computability 64
4.3.1. PCF versus S1–S9 64
4.3.2. The full abstraction problem 66
4.3.3. Intensional semantics for PCF 69
4.3.4. Other work 72
4.4. The sequentially realizable functionals 73
5. Realizability models 77
5.1. Type structures in realizability models 79
5.2. Domains in realizability models 82
6. Non-functional notions of computability. 83
6.1. Structures over $\mathbb{N}$ 83
6.2. Structures over $\mathbb{N}_\perp$ 84
6.3. A realizability perspective 86
7. Conclusion and prospectus 87
7.1. Acknowledgements 88
Appendix A. Summary of type structures 89
Appendix B. Remarks on bibliography 90

§1. Introduction.

This article is essentially a survey of fifty years of research on higher type computability. It was a great privilege to present much of this material in a series of three lectures at the Paris Logic Colloquium.

In elementary recursion theory, one begins with the question: what does it mean for an ordinary first order function on ℕ to be "computable"? As is well known, many different approaches to defining a notion of computable function — via Turing machines, lambda calculus, recursion equations, Markov algorithms, flowcharts, etc. — lead to essentially the same answer, namely the class of (total or partial) *recursive* functions. Indeed, *Church's thesis* proposes that for functions from ℕ to ℕ we identify the informal notion of an "effectively computable" function with the precise mathematical notion of a recursive function.

An important point here is that many *prima facie* independent mathematical constructions lead to the same class of functions. Whilst one can argue over whether this is good evidence that the recursive functions include *all* effectively computable functions (see Odifreddi [1989] for a discussion), it is certainly good evidence that they represent a mathematically natural and robust class of functions. And since no other serious contenders for a class of effectively computable functions are known, most of us are happy to accept Church's thesis most of the time.

Now suppose we consider second order functions which map first order functions to natural numbers (say), and then third order functions which map second order functions to natural numbers, and so on. We will use the word *functional* to mean a function that takes functions of some kind as arguments. We may now ask: what might it mean at these higher types for a functional to be "computable"? (Some reasons why we might want to ask this will be discussed shortly.)

A moment's reflection shows that a host of choices confront us if we wish to formulate a definition of higher type computability. For example:[1]

- **Domain of definition.** Do we wish to consider *partial* or *total* computable functionals? Do we want them to act on partial functions of the next type down, or just on total functions? Should they act only on the "computable" functions of this type, or on some wider class of functions?
- **Representation of functions.** If we wish to perform "computations" on functions, how do we regard the functions as given to us? As infinite graphs? As algorithms or "programs" of some kind? Or as oracles or "black boxes", for which we only have access to the input/output behaviour?
- **Protocol for computation.** What ways of interacting with functions do we allow in computations? For example, do we insist that calls to func-

[1]Many of these points are also made in a survey article by Cook (Cook [1990]), whose point of view is very close to our own.

tions are performed sequentially, or do we allow parallel function calls? Do we insist that terminating computations are in some sense finite mathematical objects, as must be the case if we are seeking a genuinely effective notion of computability — or do we allow infinite computations in accordance with the infinitistic nature of the arguments?
- **Extensionality.** Do we want to restrict our attention to computable *functions* (as implicitly assumed in the preceding discussion)? Or do we want to consider computability for other, possibly non-extensional, operations of higher type? If the latter, what do we mean by an "operation"?

The spirit in which we are asking these questions is not to demand definitive answers to them, but to make the point that many choices are possible. Indeed, as we shall see, many different responses to the above questions are exemplified by the definitions of higher type computability that have been proposed in the literature. Moreover, the effects of all these choices escalate rapidly as we climb up the types. For example, if two definitions yield different classes of computable functions of type σ, it may be difficult even to compare these definitions at type $\sigma \to \mathbb{N}$, since the domains of the functions may differ. Indeed, we often find that a question needs to have a positive answer at type σ in order to be even meaningful at type $\sigma \to \mathbb{N}$.

It thus appears that very many approaches to defining higher type computability are possible, but it is not obvious *a priori* whether some approaches are more sensible than others, or which approaches lead to equivalent notions of computability. Moreover, in contrast to the first order situation, there does not seem to be a clear canonical pre-formal notion of "effective computability at higher types" to which we can refer for guidance. (This is hardly surprising, in view of the fact that there are several possible pre-formal conceptions of what a function is.) In short, it is unclear in advance whether at higher types there is really just one natural notion of computability (as in ordinary recursion theory), or several, or indeed no really natural notions at all.

This paper is the first of a planned series of three articles devoted to the conceptual problem of finding good, natural notions of higher type computability. Whereas previous work in the area has explored various *particular* notions of computability in some detail, our wish is to take a step back and look at the overall picture. Our main objectives are as follows:

- To discover what natural notions of computability exist at higher types, and to collect evidence for their naturalness.
- To develop some basic "recursion theory" for each of these notions, analogous to the elementary parts of ordinary recursion theory.
- To investigate how these notions of computability are related.
- To provide a coherent framework for pulling together and organizing the existing knowledge in the area.

We will be concerned mainly with objects of *finite type* (that is, nth order operations for some $n \in \omega$). In principle one can also consider *transfinite* types, though we will touch on these only occasionally.

Many ideas and results relevant to our project are already known, although they are rather widely scattered across the literature in recursion theory, constructive logic and computer science, and have never previously been presented together as contributions to a single subject. In this article we will give a fairly comprehensive survey of the work to date on different approaches to higher type computability. This will amass some raw material for our project. In two sequel papers, we will present a more systematic view of much of this material, proposing some simple general frameworks for discussing the "space of possible notions of computability", and showing that within these frameworks a reasonably cohesive picture does indeed emerge from the disparate strands.

To expand on our working philosophy a little further: It appears (to us) that *a priori* considerations are by themselves of limited use in determining what are the natural notions of higher type computability — any particular definition one can write down seems to involve some choices which might be felt to be arbitrary. We are therefore led to adopt a more empirical attitude: we can explore a range of possible definitions, and see what natural notions emerge and how they are related. Various criteria may be used to determine which notions of computability count as "natural", for instance:

- Whether they arise from some intuitively appealing informal concept of "computation".
- Whether they admit a wide range of independent mathematical characterizations — the more independent the better.
- Whether they occupy some special position within the space of possible notions of computability.

There has already been much research over the last fifty years exploring different approaches to higher type computability, and we feel the time has come for bringing this material together and trying to make sense of the big picture. In view of our goals, it is natural that we should favour an eclectic attitude — since we do not know in advance where to look for good notions of computability, we should cast our net as wide as possible and embrace the diversity of definitions that have been proposed.

Our enterprise will be justified in retrospect if it does in fact lead to a coherent and satisfying picture. We will then be in a strong position to attempt a more conceptual explication of those notions of computability which we suspect to be fundamental.

To anticipate the outcome of our project, we will argue that, for computable *functionals* at least, there is in fact a manageable handful of around six natural and robust notions of higher type computability, each with a variety of different characterizations and some pleasing intrinsic properties. Although it is possible that there are other equally natural notions of computable functional awaiting discovery, the fact that very many attempts at defining a notion of computable functional lead to one of the known notions suggests (in this author's opinion) that the current picture is probably by now reasonably complete. For non-functional notions of computability, the situation is at present much more open-ended, but we are at least able to unify much of what is currently known in a satisfying way.

1.1. Motivations. Before proceeding further, we should mention some of the reasons why computability at higher types is interesting. Besides its intrinsic mathematical and conceptual appeal, the subject lies at an intriguing juncture between several areas of mathematical logic and computer science, and has (actual or potential) connections with the following areas. For reasons of space, however, we will say relatively little about these applications in the rest of the paper, choosing to concentrate on clarifying the basic notions of the subject.

1.1.1. *Constructive logic and metamathematics.* Historically, the first applications of the ideas of higher type computability were to the metamathematics of constructive systems. Computable objects of finite type can often be used to give interpretations of logics — such as *realizability* interpretations — that endow formulae with some kind of constructive content. For instance, we might stipulate that a *realizer* for a formula $\phi \Rightarrow \psi$ is a computable function mapping realizers for ϕ to realizers for ψ; in this case, formulae with nested implications will naturally lead us to consider higher type objects.

On a technical level, such interpretations can be used to obtain consistency and independence results for constructive logics. On a conceptual level, they can be helpful for clarifying various constructive views of mathematics, often from a classical standpoint. A good early discussion of possible applications of this kind appears in Kreisel [1959, Sections 1,2]. The area was extensively developed by Troelstra and his school (Troelstra [1973]), and by Beeson (Beeson [1985]), who focused on realizability and related interpretations.

In a somewhat similar spirit is Feferman's use of computable higher type objects in connection with systems for explicit mathematics and theories of finite type (Feferman [1975], [1977b]). These systems are typically intended to reflect "semi-constructivist" standpoints that suffice for most of mathematical practice. For other recent applications of this kind, see Kohlenbach [2002].

1.1.2. *Descriptive and admissible set theory.* Logical quantifiers may be regarded as objects of higher type: for instance, existential quantification over the natural numbers can be seen as a (non-computable) object ${}^2\exists\colon 2^{\mathbb{N}} \to 2$, where $2 = \{0, 1\}$. There are interesting relationships between computability *relative to* such quantifiers and logical complexity: for instance, a function on $\mathbb{N}$ is computable relative to ${}^2\exists$ (in a certain sense) iff it is definable by a hyperarithmetic predicate (see Section 3.2.1 below). This aspect of higher type recursion theory was an important ingredient Kleene's early work in the area, and was developed further by Moschovakis and others (see Section 3.2).

The relationship between higher type (relative) computability and logical definability also manifests itself in connections with admissible set theory. These connections can be exploited to apply forcing techniques from set theory to the solution of degree-theoretic problems for higher types (see Sacks [1990], and Section 3.2.2 below). Furthermore, a natural generalization of certain ideas from higher type computability leads to a good notion of "computability" on arbitrary sets, closely related to Gödel's notion of constructibility in set theory (see Normann [1978b], and Section 3.2.4 below).

1.1.3. *Abstract computability theories.* There is an enormous literature on finding suitable notions of computability for various kinds of mathematical objects, such as rings, fields, topological spaces, Banach spaces, or ordinals. (Griffor [1999] provides a good starting-point for references on these topics.) In view of this, it is not surprising that attempts have been made to develop *abstract* theories that offer a uniform account of "computability" for a wide range of structures (see *e.g.*, Moschovakis [1969], Friedman [1971], Tucker [1980]). Several of these approaches are clearly described in Hinman [1999]. Some approaches to abstract computabiliy can themselves be seen as instances of a more general theory of inductive definability (Aczel [1977]).

Higher type computability, and especially Kleene's early work, has inspired many of the ideas in these abstract approaches, and has played a useful role as a motivating example. In turn, these abstract approaches have then suggested simpler, clearer ways of presenting higher type computability. Both directions of influence may be discerned in the work of Moschovakis (see Moschovakis [1974a], [1983], [1989]).

1.1.4. *Semantics and logic of programming languages.* Ideas from higher type computability have inspired the design of modern functional programming languages. This started with the theoretical work of Scott and Plotkin (Scott [1969], Plotkin [1977]), which led eventually to the design of fully fledged programming languages such as Standard ML (Milner, Tofte, Harper, and MacQueen [1997]).

In addition, much work in theoretical computer science has been concerned with the *semantics* of programming languages. Finding a well-matched semantic model for a programming language is often tantamount to finding a good mathematical characterization of the notion of computable operation it embodies. As argued in Longley and Plotkin [1997], Longley [1999a], this can often help us to design a good logic for proving properties of programs in the language. The finite types over the natural numbers are a good target for study here, because many other computational datatypes of importance can be obtained easily as *retracts* of these types (this will be explained in Part II).

Ideas from higher type computability also play a role in Feferman's approach to computation on abstract datatypes (see *e.g.*, Feferman [1996]). In the longer term, we expect that an understanding of higher type computability will contribute particularly to the study of object oriented languages, which naturally support higher order styles of programming.

1.1.5. *Subrecursion and complexity.* Notions of higher type computation can also be used to study the computational complexity of ordinary first order functions. Many interesting *subrecursive* notions of first order computability (that is, notions that do not allow the computation of all general recursive functions) can be conveniently characterized via systems for higher type recursion. For example, the ϵ_0-*recursive* functions (*i.e.*, the provably total functions of Peano arithmetic) are precisely those definable in Gödel's System T (see Section 3.1). Furthermore, an ordinal stratification of these functions, corresponding to the *extended Grzegorczyk hierarchy*, can be defined very elegantly using some simple higher type functionals (see Schwichtenberg [1975], [1999]). As argued in Schwichtenberg [1999], it seems that in giving definitions of functions there is some inherent connection or trade-off between type complexity and ordinal complexity.

Recently, characterizations of this kind have been achieved for much lower complexity classes, including even the polytime computable functions (see Bellantoni, Niggl, and Schwichtenberg [2000] and the papers cited therein).

In addition, of course, one can look for natural complexity classes at higher type. Ideas from higher type computability have informed the definition of complexity-theoretic notions (see *e.g.*, Cook [1990]), but it seems that many of the basic notions of higher type complexity are still open to discussion. Indeed, in order to formulate a notion of feasible computation at higher types (for instance), we will have to confront all the choices mentioned earlier, and many others besides. Clearly, a good understanding of higher type computability will stand us in good stead if we wish to clarify the fundamental notions of complexity at higher types.

The current state of the art regarding feasible computation at higher types is described in detail in Irwin, Kapron, and Royer [2001a], [2001b]. Other subrecursive notions of computability at higher types have been considered in Schwichtenberg [1991], Niggl [1993].

1.1.6. *Real number computability.* The connection between computability over the reals and higher type computability was recognized as far back as Lacombe [1955a], [1955b], [1955c] and Grzegorczyk [1957], and later manifested itself in the context of constructive interpretations of systems for analysis (*e.g.*, in Troelstra [1973]). Notions of computability over the reals and other metric spaces underpin constructive recursive analysis of the Markov school on the one hand (see Aberth [1980], Beeson [1985]), and classical computable analysis on the other (see Pour-El and Richards [1989], Weihrauch [2000b], Bauer [2000]). Interest in real number computability has also recently been awakened within computer science (see Escardó [1996]), and exact real number computation appears attractive as a potential application area for higher type programming. For example, *integration* for real functions corresponds to a second order computable operation over the reals, and hence to a third order computable operation over ℕ (see Simpson [1998]); an operator for solving differential equations might therefore involve a fourth order operation over ℕ.

Higher types over the reals, and some associated notions of computability, have recently been considered *e.g.*, in Normann [2001], [2002], Bauer, Escardó, and Simpson [2002], Korovina and Kudinov [2001]. As with higher type complexity, it appears that the natural notions here are still open to discussion — though once again, a good understanding of the higher types over ℕ will surely help.

1.1.7. *Computability in physics.* Notions of computability in analysis can in turn be applied to questions of computability in various physical theories. Some of this territory is explored in Pour-El and Richards [1989] (see also Pour-El [1999]). Besides providing a foundation for this work, it seems conceivable that the study of higher type computability might suggest alternative definitions of computability for physical systems which may hold some philosophical interest. For some intriguing speculations along these lines, see Cooper [1999].

1.2. Overview of the series. This series of articles is intended as a fairly comprehensive account of what is currently known about notions of higher type computability over the natural numbers.

The present Part I is a historical survey of the work to date on higher type computability, tracing the various strands of research which have contributed to our present understanding. This is intended to serve several purposes: firstly, to offer a gentle introduction to the main ideas of the subject; secondly, to document the genesis of these ideas; thirdly, to facilitate comparison between different strands of work by placing them side by side in a uniform setting; and fourthly, to provide a reasonably complete map of the rather bewildering literature in the area. We have tried to include just enough technical detail to make the mathematical substance intelligible, without losing the broad sweep of the story. The point of view we wish to advocate is that all the strands of research that we describe can naturally be seen, with hindsight, as contributions to a single coherent subject. In the remaining papers in the series, we will attempt a more systematic and technically detailed exposition of this subject.

In Part II we will try to organize the material concerning notions of computable *functional* (that is, extensional operations of higher type), and the relationships between them. Here we will work within a general framework given by some simple definitions involving *finite type structures*. Although this framework is very simple and even somewhat crude, it suffices for clarifying much of the existing material. After developing the necessary general concepts, we will consider in turn the various good notions of *total* and *partial* computable functional. In both the total and partial settings, we will present arguments for the impossibility of a "Church's thesis for higher types". We will also discuss ways in which total and partial functionals can be related and combined.

In Part III we will consider, more generally, notions of computable *operation* (not necessarily extensional) — for example, the notions of computability embodied by various non-functional programming languages. In order to articulate these notions, we will use a more sophisticated general framework based on ideas of *realizability* — this will extend and refine the theory of Part II in a mathematically satisfying manner. Once again, we develop the general theory, then survey within this framework some of the notions of computability that appear to have some claim to naturalness.

Naturally, much of the material covered in Part I will be treated again from a different perspective in Parts II and III. We feel that this kind of overlap is justified in the interest of presenting a rounded view that takes account of both the historical and the purely logical aspects of the subject. Indeed, the historical and logical parts of the series are intended to be complementary, in the sense that each tries to emphasize ideas that receive scant attention in the other.

1.3. Outline of the present paper. As a glance at Figure 1 on page 19 will confirm, the study of higher type computability has not developed in a coherent, orderly fashion. Rather, it is mostly the result of the parallel activity of several research communities, each with their own set of motivations, and it is only in retrospect that the various strands can be seen as parts of a coherent whole. It is therefore not surprising that the history of the subject appears as somewhat chaotic.

The parallel nature of the subject's development, in particular, makes the history difficult to describe: some compromise between a strictly chronological presentation and a thematic one is necessary, and no linear ordering of the material seems completely satisfactory from an expository point of view. Here we have adopted the following course. In Section 2 we describe, more or less chronologically, the early work on computability at type 2, taking us up to about 1958. Around this time, several notions of computability at all finite types made their debut; at this point the subject effectively split into several streams, and it is only over the last few years that these have begun to converge again. For the main body of the paper (Sections 3 and 4) we therefore treat each of the main notions of higher type computability in turn, taking them (roughly) in order of their first appearance in the literature, and giving separate chronological accounts of the developments relating to each of them.

In Section 3 we discuss around four different notions of *total* computable functional, and in Section 4 we consider a similar number of notions of *partial* computable functional. (Fortunately for our scheme, all of the total notions made their first appearance before any of the partial ones!) As we shall see, the partial notions tend to be simpler and to have more pleasant properties than the total ones. Turning to more recent developments, in Section 5 we describe some ideas from realizability which cross-cut many of these streams. Finally, in Section 6 we briefly discuss some ideas relating to *non-functional* notions of "computable operation" at higher types.

For convenience, some diagrams summarizing the main structures of interest and the relationships between them have been included in an appendix.

We have tried to make our account as accurate and complete as possible. However, shortcomings are inevitable in a work of this kind, and the author can only offer his apologies to anyone whose work he has inadvertently misrepresented or overlooked. He would be glad to be informed of any significant errors or omissions so that these may be rectified in a future publication.

Throughout the paper we presuppose a good knowledge of elementary recursion theory, and some general background in logic. A few further prerequisites of a more specialized nature will be summarized in Section 1.5.

1.4. Notation. We first fix some general notational conventions. For any set X, we write $X_\perp$ for the set $X \sqcup \{\perp\}$, where $\perp$ is some distinguished element which intuitively will represent the *non-termination* of a process which is attempting to compute a value in X. We write $f : X \rightharpoonup Y$ to mean "f is a partial function from X to Y". Any partial function $f : X \rightharpoonup Y$ may be *represented* by a total function $X \to Y_\perp$, though formally we shall distinguish between the two entities.

We will use the following notational conventions in connection with potentially non-denoting expressions e, e' arising from the use of partial functions.

- $e\downarrow$ means "the value of e is defined" (that is, e denotes something).
- $e\uparrow$ means "the value of e is not defined".
- $e = e'$ means "the values of e and e' are both defined and they are equal" (strict equality).
- $e \simeq e'$ means "if either e or e' is defined then so is the other and their values are equal" (Kleene equality).

Note that these conventions relate to the definedness of mathematical expressions rather than the termination of computations. In particular, we have $\perp\downarrow$.

If e is an expression possibly involving the variable x, we write $\Lambda x.e$ to mean the set-theoretic (total or partial) function that maps x to e; the intended domain of this function will be determined by the context. Thus, the expression $\Lambda x.e$ names the function defined in more standard notation by $x \mapsto e$. However, the Λ notation seems more convenient in complex expressions, and it also provides a semantic counterpart to the formal syntax of the λ-calculus which we shall frequently use (see Section 1.5.2).

We write $\mathbb{N}$ for the set of natural numbers including 0. We also write:

- $\mathbb{N}^{\mathbb{N}}$ for the set of all (set-theoretic) total functions from $\mathbb{N}$ to $\mathbb{N}$,
- $\mathbb{N}^{\mathbb{N}}_{p}$ for the set of all partial functions from $\mathbb{N}$ to $\mathbb{N}$,
- $\mathbb{N}^{\mathbb{N}}_{rec}$ for the set of total recursive functions from $\mathbb{N}$ to $\mathbb{N}$,
- $\mathbb{N}^{\mathbb{N}}_{p\,rec}$ for the set of partial recursive functions from $\mathbb{N}$ to $\mathbb{N}$.

We write $\mathrm{Seq}(X)$ for the set of finite sequences over a set X; we will use the notation $[x_1, \ldots, x_n]$ to display such sequences. We will suppose $\langle - \rangle \colon \mathrm{Seq}(\mathbb{N}) \to \mathbb{N}$ is some fixed effective coding for finite sequences, and write $\langle x_1, \ldots, x_n \rangle$ in place of $\langle [x_1, \ldots, x_n] \rangle$. Given $f : \mathbb{N} \rightharpoonup \mathbb{N}$, we define its *course-of-values* function $\widetilde{f} : \mathbb{N} \rightharpoonup \mathbb{N}$ by

$$\widetilde{f}(n) \simeq \langle f(0), \ldots, f(n-1) \rangle.$$

We also suppose we have some effective indexing scheme for the partial recursive functions, given for example by an effective enumeration of Turing machines. We will write ϕ_m for the partial recursive function from $\mathbb{N}$ to $\mathbb{N}$ with recursive index m.

If X is any set and R is a partial equivalence relation (that is, a symmetric, transitive relation) on X, we write X/R for the set of R-equivalence classes over X.

We will follow certain conventions regarding variables and also in our use of certain typefaces. Ordinary mathematical fonts will be used for variables of all kinds; generally speaking, we will use

- i, j, k, l, m, n, r to range over $\mathbb{N}$;
- α, β to range over $\mathrm{Seq}(\mathbb{N})$;
- f, g, h to range over first order functions, or sometimes over functions of arbitrary type;
- F, G to range over second order objects (functions or operations);
- Φ, Ψ, Θ to range over higher order objects (*i.e.*, third order or above).

We will also use subscripted and superscripted variants of these symbols in the same way. We will occasionally depart from the above conventions when it is convenient to do so. Other variables (*e.g.*, x, y, z) will be used more flexibly as we have need of them.

We use boldface letters as abbreviations for *vectors*, or lists of variables. More precisely, a symbol such as $\boldsymbol{x}$, wherever it occurs, will textually abbreviate either $x_1 \ldots x_{l_x}$ or $x_1, \ldots, x_{l_x}$ (as demanded by the context), where $l_x \geq 0$.

We use Roman boldface (*e.g.*, ***Set***) for the names of particular categories of interest, and uppercase sans serif font (*e.g.*, HEO) for the names of particular type structures (see Section 1.5.1).

A few other typographical conventions associated with λ-calculi and types will be introduced in the course of the next section.

1.5. Prerequisites. Throughout this paper we will make incessant use of the idea of a *type structure*; we will also make reference to the *λ-calculus* and occasionally to *cartesian closed categories*. These notions will enormously facilitate our task of giving a unified presentation of our material. Our use of these concepts will sometimes mean recasting original definitions into a rather more modern form, but we believe this will not do too much violence to the historical point of view.

We now review the material that we shall need on these topics.

1.5.1. *Types and type structures.* The following basic concepts are central to the entire paper and will be used ubiquitously.

Given any set Γ of *basic type symbols* γ, the (*finite* or *simple*) *types* over Γ are the formal expressions σ built up according to the following grammar:

$$\sigma ::= \gamma \mid (\sigma_1 \to \sigma_2)$$

Informally, each symbol $\gamma \in \Gamma$ will represent some set of basic entities, and $(\sigma_1 \to \sigma_2)$ will represent the type of functions from σ_1 to σ_2. We use ρ, σ, τ as variables ranging over types. We will omit brackets wherever possible, and regard $\to$ as right-associative, so that $\rho \to \sigma \to \tau$ means $(\rho \to (\sigma \to \tau))$. We define the *level* of a type inductively by

$$\mathit{level}\,(\gamma) = 0, \quad \mathit{level}\,(\sigma \to \tau) = \max(1 + \mathit{level}\,(\sigma), \mathit{level}\,(\tau)).$$

Except where otherwise stated, we will be considering finite types over the single basic type symbol $\overline{0}$, which usually represents the type of natural numbers. We then define the *pure types* $\overline{n}$ inductively by $\overline{n+1} = \overline{n} \to \overline{0}$. If (X_σ) is a family of mathematical objects or relations indexed by types σ, we will often write just X_n in place of $X_{\overline{n}}$.

Our fundamental notion will be that of a *type structure*. For the purpose of the present paper, the following definition will suffice:

Definition 1.1 (Type structures).

(i) *A* partial type structure *A will consist of a family of sets A_σ (one for each type), together with "application" functions $\cdot_{\sigma\tau} : A_{\sigma\to\tau} \times A_\sigma \rightharpoonup A_\tau$.*

(ii) *A* total type structure, *or more simply a* type structure, *is a partial type structure in which all the application functions are total.*

We usually omit the type subscripts from the application functions, and treat $\cdot$ as a left-associative infix, so that $f \cdot x \cdot y$ means $(f \cdot x) \cdot y$. In accordance with the convention mentioned above, we often write A_n in place of $A_{\overline{n}}$. By a *type n object* of A we will mean an element of some A_σ where $\mathit{level}\,(\sigma) = n$. We say A is a (partial or total) type structure *over X* if $A_0 = X$.

Most of the examples we consider will be type structures over $\mathbb{N}$ or $\mathbb{N}_\perp$. An important example is the *full set-theoretic* type structure S over $\mathbb{N}$, in which $\mathsf{S}_0 = \mathbb{N}$ and $\mathsf{S}_{\sigma\to\tau}$ is the classical set of all functions from S_σ to S_τ.

A partial type structure A is *extensional* if for all types σ, τ and all $f, g \in A_{\sigma\to\tau}$ we have

$$(\forall x \in A_\sigma.\, f \cdot x \simeq g \cdot x) \Longrightarrow f = g.$$

It is easy to see that any extensional [partial] type structure A is isomorphic to a [partial] type structure B in which each set $B_{\sigma\to\tau}$ is a set of [partial] functions from B_σ to B_τ. We will therefore often refer to objects of type level 2 or more in extensional partial type structures as *functionals*.

There is a standard way to obtain extensional type structures from an arbitrary type structure:

Definition 1.2 (Extensional collapse). *Let A be any partial type structure over X.*

(i) *Given any partial equivalence relation $\approx$ on X, define a partial equivalence relation $\approx_\sigma$ on each A_σ as follows:*
- *$x \approx_0 y$ iff $x \approx y$;*
- *$f \approx_{\sigma\to\tau} g$ iff for all $x, y \in A_\sigma$, $x \approx_\sigma y$ implies $f \cdot x\downarrow$, $g \cdot y\downarrow$ and $f \cdot x \approx_\tau g \cdot y$.*

Now let $\mathsf{EC}(A, \approx)$, *the* extensional collapse *of A with respect to $\approx$, be the total extensional type structure defined by* $\mathsf{EC}(A, \approx)_\sigma = A_\sigma / \approx_\sigma$, *with application inherited from A.*

(ii) *Define* $\mathsf{EC}(A)$, *the* extensional collapse *of A, to be* $\mathsf{EC}(A, =)$.

We will often speak, somewhat informally, of an element of a type structure A being *hereditarily P* for some property P. What we typically mean by this may be explained reasonably precisely by induction on types as follows:

- An element $x \in A_0$ is hereditarily P iff x has property P;
- An element $f \in A_{\sigma\to\tau}$ is hereditarily P if the restriction f' of f to the hereditarily P elements $x \in A_\sigma$ has property P, and for all such x, $f \cdot x$ is hereditarily P.

For example, if A is a type structure over $\mathbb{N}_\perp$, a hereditarily total functional of type $\overline{2}$ is a functional that acts totally on functions that act totally on elements of $\mathbb{N}$. Clearly, if all the elements of A have some property P, then they are all hereditarily P.

A more comprehensive armoury of definitions relating to type structures will be presented in Part II.

1.5.2. *The λ-calculus.* Next we review the basics of untyped and simply typed λ-calculi. For more details, see for example Barendregt [1984].

The λ-calculus is a convenient formal language for talking about functions and application. To define an *untyped* λ-calculus, one specifies a (possibly empty) set C of *constant* symbols, and we also assume we have available an unlimited supply of *variable* symbols. We will use teletype font for particular constant symbols that we shall introduce (*e.g.*, $\mathtt{0}, \mathtt{succ}$), and use c as a metavariable ranging over constant symbols. We will use ordinary Roman letters $x, y, z, f, g, h, \ldots$ as variable symbols. As usual in logic, we will not bother to distinguish notationally between particular variables and metavariables ranging over variable symbols.

The *untyped λ-terms U* over C are then built up according to the following grammar:

$$U ::= c \mid x \mid (\lambda x.U_1) \mid (U_1 U_2)$$

We use U, V, W as metavariables ranging over λ-terms. We will omit brackets wherever possible, taking application to be left-associative, so that UVW means $((UV)W)$. We will also write $\lambda x_1 \ldots x_n.U$ as an abbreviation for $\lambda x_1.\cdots.\lambda x_n.U$.

Informally we may read $U_1 U_2$ as "U_1 applied to U_2", and $\lambda x.U_1$ as "the function mapping any element x to U_1". We view λx as a *binder* analogous to the quantifiers $\forall x$ and $\exists x$ in logic; thus we have evident notions of free and bound variable occurrences, closed terms, and substitution. We write $U[V/x]$ for the result of substituting V for all free occurrences of x in U, renaming bound variables if necessary to avoid capture.

The untyped λ-calculus is a very fluid system: any term can be applied to any other term and even to itself. Often we wish to consider more restricted systems in which natural type distinctions between functions and arguments are enforced. For our purposes it will suffice to consider *simply typed* λ-calculi, in which the types σ are given precisely as in Section 1.5.1 above. To define a simply typed λ-calculus, one specifies a set C of constant symbols c^σ, each decorated with some type σ; we also assume we have an infinite supply of variable symbols x^σ for each type σ. We first define the set of untyped λ-terms U as above; we then define a relation $U : \sigma$ (read as "U is of type σ") inductively by means of the following clauses:

- $c^\sigma : \sigma$ for any constant c^σ,
- $x^\sigma : \sigma$ for any variable x^σ,
- if $U : \sigma \to \tau$ and $V : \sigma$ then $UV : \tau$,
- for any variable x^σ, if $U : \tau$ then $\lambda x^\sigma.U : \sigma \to \tau$.

We will frequently omit type superscripts where these can be inferred from the context. We say U is *well-typed*, or is a *simply typed λ-term*, if $U : \sigma$ for some (necessarily unique) σ.

We can think of a λ-calculus (whether untyped or simply typed) not just as a formal notation for functions, but also as a kind of programming language in which one can perform computations by symbolic manipulation. For instance, one may define a *reduction* relation $U \rightsquigarrow V$ on the terms of a λ-calculus, intended to capture the idea of a single computation step. The definition of $\rightsquigarrow$ will depend on the λ-calculus in question, but (for the purposes of this paper) it will always include at least the *β-rule*:

$$(\lambda x.U)V \rightsquigarrow U[V/x]$$

Frequently, the definition will also include the following *congruence rules*:

$$\text{if } U \rightsquigarrow U', \text{ then } UV \rightsquigarrow U'V,\ WU \rightsquigarrow WU', \text{ and } \lambda x.U \rightsquigarrow \lambda x.U'.$$

In addition, there may be other special reduction rules involving the constants of the language in question. In general we will write $\rightsquigarrow^*$ for the reflexive transitive closure of $\rightsquigarrow$. A term U is a *final value* with respect to $\rightsquigarrow$ if there is no V such that $U \rightsquigarrow V$. We write $U \Downarrow V$ if $U \rightsquigarrow^* V$ and V is a final value.

Simply typed λ-calculi provide good languages for defining objects of finite type. An *interpretation* of a simply typed λ-calculus $\mathcal{L}$ in a type structure A is a function assigning to each closed term $U : \sigma$ of $\mathcal{L}$ an element $[\![U]\!] \in A_\sigma$, in such a way that $U \rightsquigarrow V$ implies $[\![U]\!] = [\![V]\!]$. A λ-calculus that admits an interpretation in A is often a very convenient formal language for denoting elements of A. We may say an element x of A is $\mathcal{L}$*-definable* (for a particular language $\mathcal{L}$) if $x = [\![U]\!]$ for some closed term U of $\mathcal{L}$.

A *theory* on $\mathcal{L}$ will (for our purposes) be a type-respecting equivalence relation $\sim$ on closed terms of $\mathcal{L}$ which includes the reduction relation and which is a congruence with respect to application. Any interpretation $[\![-]\!]$ of $\mathcal{L}$ induces a theory on $\mathcal{L}$, given by $U \sim V$ iff $[\![U]\!] = [\![V]\!]$.

The set of closed terms of a simply typed λ-calculus $\mathcal{L}$ is itself a type structure, with application given by juxtaposition of terms. Furthermore, the quotient of this type structure modulo any theory on $\mathcal{L}$ is again a type structure. Type structures obtained from theories in this way are called *(closed) term models* for $\mathcal{L}$. Interesting term models can often be obtained by identifying terms that have equivalent behaviour in some sense.

1.5.3. *Cartesian closed categories.* We now give a brief sketch of the notion of a cartesian closed category. Only a general impression will be required in this paper. Further details may be found in Lambek and Scott [1986].

Informally, a cartesian closed category is one in which for any objects X and Y, we have an object Y^X playing the role of the space of functions from X to Y. A helpful motivating example is the classical category ***Set*** of sets and functions, in which Y^X is simply the set of all functions from X to Y. Note that for any set Z, functions from $Z \times X$ to Y correspond precisely to functions from Z to Y^X. Abstracting the essential features of this situation leads to the following definition:

Definition 1.3 (Cartesian closed categories). *A* cartesian closed category *is a category $\mathcal{C}$ with finite products (including a terminal object 1), in which for any objects X and Y we have an object Y^X and a morphism $\epsilon_{XY} : Y^X \times X \to Y$ with the following property: for all morphisms $f : Z \times X \to Y$ there is a unique morphism $\overline{f} : Z \to Y^X$ such that*

$$f = \epsilon_{XY} \circ \langle \overline{f}, \mathrm{id}_X \rangle.$$

It is a consequence of the definition that the object Y^X is always uniquely determined up to isomorphism. The reader unfamiliar with the above definition should satisfy himself that in the case of ***Set*** it does indeed characterize the set of all functions from X to Y up to a canonical bijection.

Given any object X in a cartesian closed category $\mathcal{C}$, we may obtain a type structure as follows. First define an interpretation $[\![-]\!]$ of the finite types as objects of $\mathcal{C}$:

$$[\![\overline{0}]\!] = X, \quad [\![\sigma \to \tau]\!] = [\![\tau]\!]^{[\![\sigma]\!]}$$

Now define a type structure $A = \mathsf{T}(\mathcal{C}, X)$ by taking $A_\sigma = \mathrm{Hom}(1, [\![\sigma]\!])$, with $\cdot_{\sigma\tau}$ the evident function induced by ϵ_{XY}. Many type structures of interest arise in this way from mathematically natural categories. Clearly, if $\mathcal{C} =$ ***Set*** and $X = \mathbb{N}$, this construction yields the full set-theoretic type structure S.

A cartesian closed category is *well-pointed* when for all $f, g \colon X \to Y$, if $f \circ x = g \circ x$ for every $x \colon 1 \to X$ then $f = g$. If $\mathcal{C}$ is well-pointed, then all type structures $\mathsf{T}(\mathcal{C}, X)$ will be extensional.

There is a very close relationship between cartesian closed categories and the simply typed λ-calculus. For instance, given a λ-calculus $\mathcal{L}$ and a suitable interpretation of its basic types and constant symbols in a cartesian closed category $\mathcal{C}$, one can define an interpretation of $\mathcal{L}$ in $\mathcal{C}$, in which a λ-term $U : \tau$ with free variables $x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}$ is interpreted by a morphism

$$[\![U]\!] \colon [\![\sigma_1]\!] \times \cdots \times [\![\sigma_n]\!] \to [\![\tau]\!].$$

In the case of a single ground type $\overline{0}$, such an interpretation clearly gives rise to an interpretation of $\mathcal{L}$ in $\mathsf{T}(\mathcal{C}, [\![\overline{0}]\!])$ in the sense of Section 1.5.2.

We may say $\mathcal{C}$ is a *model* for $\mathcal{L}$ if we have an interpretation such that $U \rightsquigarrow V$ implies $[\![U]\!] = [\![V]\!]$. In fact, the β-rule and congruence rules mentioned in Section 1.5.2 are automatically validated by any interpretation of $\mathcal{L}$ in a cartesian closed category.

We will sometimes refer in passing to the notion of a *topos*. All that the reader will need to know is that a topos is a cartesian closed category with some strong additional properties, giving rise to a very rich categorical structure. In fact, any topos provides a model for higher order intuitionistic logic, and can therefore be viewed as a kind of "universe" for much of intuitionistic mathematics. The leading example of a topos is ***Set***; in this case, the corresponding interpretation of logic coincides with the familiar classical one. Again, more information can be found in Lambek and Scott [1986].

1.5.4. *Historical remarks.* The concept of functions of arbitrary finite type (and even of transfinite type) appeared in Hilbert [1925], in a discussion of Cantorian set theory. The language of simple types, and the simply typed λ-calculus, were introduced in Church [1940], though the question of interpretations of the system was deliberately left open. Definitions that more or less resemble our notion of type structure have appeared very many times in the literature: the first such was given in Henkin [1950], who considered interpretations of Church's system in the type structure S.

Particular instances of the extensional collapse construction (in the form in which we have defined it) appear in Kreisel [1959] and Kleene [1959a]. The general construction was probably folklore from an early stage, though its debut in the literature seems to be in Zucker [1971].

The connections between λ-calculi and cartesian closed categories were established in the 1970s by Lambek and others (see Lambek and Scott [1986]).

§2. Early work: Computability at type 2.

2.1. Prehistory. The main ideas concerning computability for type 1 functions of course date back to the development of basic recursion theory in the 1930s (Gödel [1931], Church [1936], Turing [1937b], [1937a], Kleene [1936a], [1936b], Post [1936]). These early papers furnished several characterizations of the class of (partial) recursive functions. The first explicit formulation of Church's thesis appears in Church [1936]. In Turing [1939, §4] Turing introduced the notion of a computing machine equipped with an oracle for deciding non-computable properties, but considered this only as a means of defining first order computability relative to a *fixed* oracle, so cannot truly be said to have introduced the concept of a computable type 2 function.

2.2. Banach-Mazur functionals. A very early definition of a class of type 2 functionals involving a notion of computability is due to Banach and Mazur (Banach and Mazur [1937]) (see also Mazur [1963]):

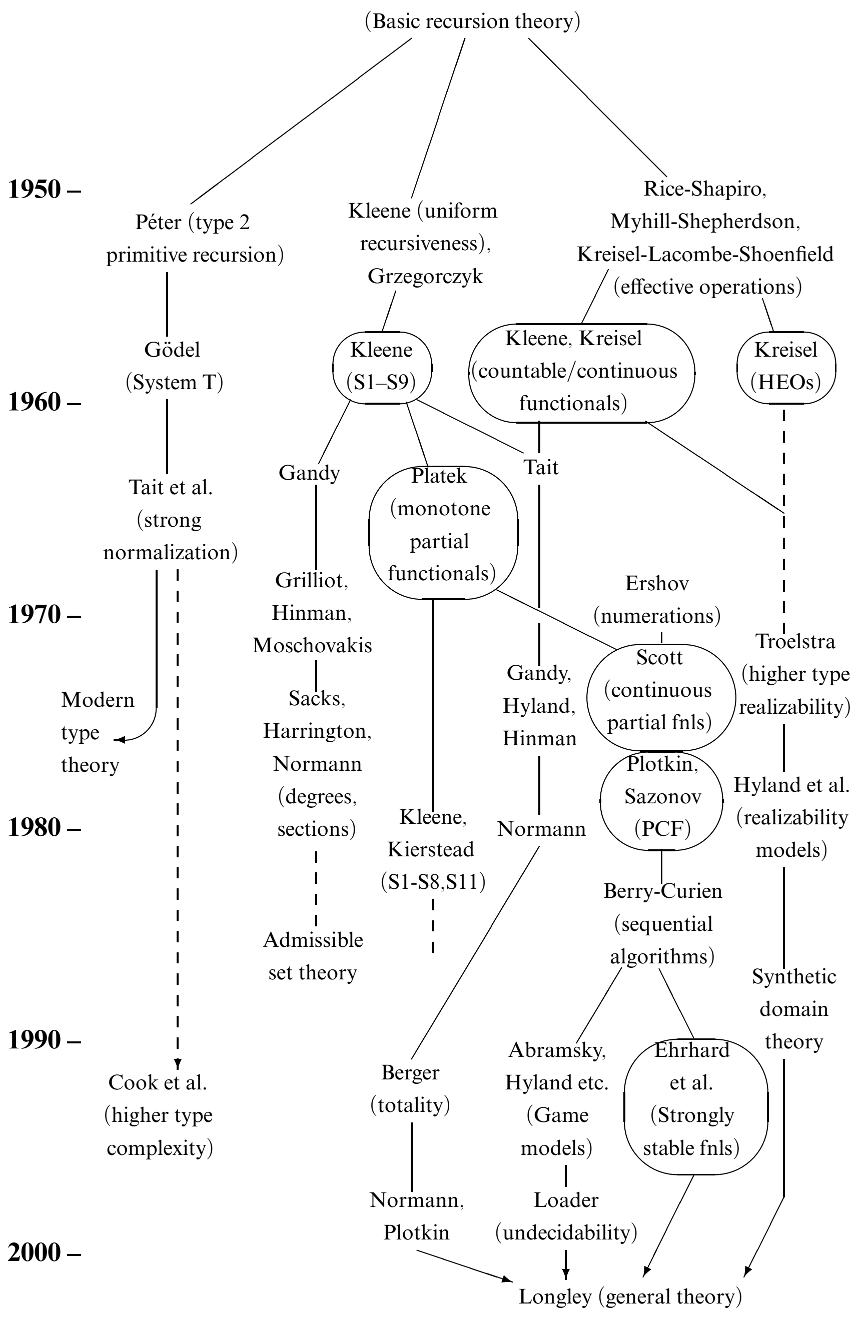
Definition 2.1. *A total function $F : \mathbb{N}^{\mathbb{N}}_{rec} \to \mathbb{N}$ is* Banach-Mazur *if, for every total recursive function $h : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, the function $\Lambda x.F(\Lambda y.h(x,y)) : \mathbb{N} \to \mathbb{N}$ is total recursive.*

Notice that this condition says that, in some sense, F carries computable functions to computable functions, but it does not tell us how given g one might compute $F(g)$ in any sense. For this reason, the notion is rather tangential to the story we tell here — we do not regard it as a genuine candidate for a notion of computable functional, but rather as a property which computable functionals may possess. Early results showed that every computable functional (in the senses discussed below) is Banach-Mazur, but not *vice versa* (see Friedberg [1958a]), and some relationships to other properties of functionals were considered in Pour-El [1960]. Most of this material is helpfully summarized in Rogers [1967, §15.3].

The Banach-Mazur functionals later reappeared in the work of Lawvere and Mulry (Mulry [1982]), whose *recursive topos* provides a natural generalization of the notion to higher types (see also Section 4.2.4).

2.3. Computations on pure functions. The first explicit definition of a genuine notion of type 2 computability, as far as we are aware, was given by

FIGURE 1. History of higher type computability: a selective outline.


(Basic recursion theory)
1950 –
1960 –
1970 –
1980 –
1990 –
2000 –
Péter (type 2 primitive recursion)
Kleene (uniform recursiveness), Grzegorczyk
Rice-Shapiro, Myhill-Shepherdson, Kreisel-Lacombe-Shoenfield (effective operations)
Gödel (System T)
Kleene (S1–S9)
Kleene, Kreisel (countable/continuous functionals)
Kreisel (HEOs)
Tait et al. (strong normalization)
Gandy
Platek (monotone partial functionals)
Tait
Modern type theory
Grilliot, Hinman, Moschovakis
Ershov (numerations)
Troelstra (higher type realizability)
Sacks, Harrington, Normann (degrees, sections)
Gandy, Hyland, Hinman
Scott (continuous partial fnls)
Plotkin, Sazonov (PCF)
Hyland et al. (realizability models)
Kleene, Kierstead (S1-S8,S11)
Normann
Berry-Curien (sequential algorithms)
Admissible set theory
Synthetic domain theory
Cook et al. (higher type complexity)
Berger (totality)
Abramsky, Hyland etc. (Game models)
Ehrhard et al. (Strongly stable fnls)
Normann, Plotkin
Loader (undecidability)
Longley (general theory)

Péter in Péter [1951a], [1951b] (although a somewhat similar definition had been sketched in Hilbert [1925]). Péter considered a schema for "primitive recursion of the second degree" as a means of defining total functions with arguments of type $\mathbb{N}^{\mathbb{N}}$ as well as $\mathbb{N}$. Sacrificing some generality for the sake of clarity, the basic idea is as follows: given previously defined functionals G, H of suitable types, one may construct a new function $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that

$$\begin{aligned} F(0, m) &= G(m), \\ F(n+1, m) &= H(\Lambda y.F(n, y), n, m). \end{aligned}$$

By allowing additional parameters $g_1, \dots, g_r$ of type $\mathbb{N}^{\mathbb{N}}$, we are able to construct new type 2 functionals from old ones. We may then define a class of *type 2 primitive recursive* functionals by starting from a suitable set of "basic" functionals and closing under substitution and primitive recursions of the above kind.

Péter showed that a certain class of "transfinite recursions" at type 1 could be systematically replaced by primitive recursions at type 2. Thus, for instance, the well-known *Ackermann* function, though not primitive recursive in the usual sense, could be defined by a type 2 primitive recursion.

In his famous book (Kleene [1952]), Kleene gave schemata for defining primitive, total and partial recursive functions *uniformly in* a finite list of functions $g_1, \dots, g_l \in \mathbb{N}^{\mathbb{N}}$ (§47, §58, §63). Restricting for simplicity to the case $l = 1$, the definitions are as follows.

DEFINITION 2.2 (Uniform computability).

(i) *A function* $f : \mathbb{N}^k \to \mathbb{N}$ *is* primitive recursive uniformly in g *if* f *can be defined from* g (*together with the usual repertoire of basic functions*) *by means of composition and* (*ordinary first order*) *primitive recursion.*

(ii) $f : \mathbb{N}^k \rightharpoonup \mathbb{N}$ *is* partial recursive uniformly in g *if* f *can be defined from* g *plus the basic functions via composition, primitive recursion and minimization. If in addition the definition of* f *results in a total function for all values of* $g \in \mathbb{N}^{\mathbb{N}}$, *we say* f *is* total recursive uniformly in g.

Kleene made explicit the possibility of regarding g as a variable and thus obtaining type 2 functionals $F : \mathbb{N}^{\mathbb{N}} \times \mathbb{N}^k \rightharpoonup \mathbb{N}$. Note that Kleene's notion of primitive recursive functional is more restrictive than Péter's, because whereas in Kleene's definition type 1 functions simply enter the computation as additional basic functions which remain fixed throughout, Péter's scheme in some sense allows infinitely many values of F to be collected into a new type 1 function during the course of a recursive computation. Kleene's definition yields just the usual class of primitive recursive functions at type 1.

Kleene also showed that his partial recursive functionals can be characterized equivalently as those computable by Turing machines with oracles (*op. cit.*, Chapter XIII).[2]

In two papers from 1954, Grzegorczyk (Grzegorczyk [1955b], [1955a]) considered a class of computable total functionals with arguments in $\mathbb{N}$ and $\mathbb{N}^{\mathbb{N}}$, defined as the smallest class containing some basic functionals and closed under substitution and minimization. It is fairly easy to see directly (and is immediate from Kleene's later results) that Grzegorczyk's notion coincides with Kleene's notion of uniform total recursiveness, though this does not seem to have been noted in the literature of the time. In Grzegorczyk [1955b], Grzegorczyk showed that his definition was equivalent to an Herbrand-Gödel style definition in terms of an equational calculus. In Grzegorczyk [1955a] he investigated some properties of this notion, showing that all computable functionals were *continuous*, and that certain *modulus of continuity* functionals were computable.

The relationship between computability and continuity was to become a recurring theme in the subject. In the case of Grzegorczyk's result (and others in the same vein), the intuition is simple: if we apply a computable type 2 function F to a type 1 function g, the "computation" of $F(g)$ can only interrogate g at finitely many arguments before terminating, so for any other function g' agreeing with g on this finite portion we would have $F(g) = F(g')$. Thus F is continuous with respect to the familiar Baire topology.

2.4. Computations on Kleene indices. All the approaches to type 2 computability mentioned above share the feature that the type 1 arguments are presented simply as oracles or "black boxes": the only way to interact with them is to feed them with an argument and observe the outcome. In parallel with this was another strand of research which considered notions of type 2 computability in which the type 1 arguments were presented as recursive indices (say, as Gödel numbers for Turing machines). Given a recursive index we can of course do everything that we can do with an oracle, since we can always apply the index to a type 0 argument. However, it would seem *a priori* that one might be able to do more with an index than with an oracle, since we are given extra *intensional* information — intuitively, we can "look inside" the box and examine how the machine or program is working.

The following definition will allow us to state the main results succinctly:

Definition 2.3 (Effective operation). *Let R be either of the sets $\mathbb{N}^{\mathbb{N}}_{rec}$ or $\mathbb{N}^{\mathbb{N}}_{p\,rec}$. A partial function $F : R \rightharpoonup \mathbb{N}$ is a* partial effective operation *on R*

[2]Kleene introduced, somewhat peripherally, a notion of partial recursiveness uniformly in a list of *partial* functions $\mathbb{N} \rightharpoonup \mathbb{N}$ (Kleene [1952, §63]), but did not study it in detail. As pointed out in Platek [1966, pp. 128–130], Kleene's particular definition has some rather undesirable features; nevertheless, it turns out to give rise to the class of parallel-computable type 2 functions, see Section 4.2.

if there exists a partial recursive function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ *such that for any* $m \in \mathbb{N}$ *with* $\phi_m \in R$ *we have* $F(\phi_m) \simeq f(m)$. *If additionally* F *is total, we say it is a* total effective operation *on* R.

Thus, an effective operation is given by a computable function f acting on recursive indices, which f may manipulate in any way it likes, subject only to the requirement of *extensionality*: f must give the same result on different indices for the same type 1 function.

In fact, all four cross-combinations of total and partial function spaces were investigated in the early literature. First, a theorem of Rice (Rice [1953]), originally phrased in terms of decidable properties of r.e. sets, essentially says the following:

Theorem 2.4 (Total acting on partial). *Every total effective operation on* $\mathbb{N}^{\mathbb{N}}_{p\,rec}$ *is constant.*

This led, via a theorem of Rice and Shapiro (Rice [1956]), to the following important result, often known as the Myhill-Shepherdson theorem. It was first obtained in Myhill and Shepherdson [1955], and independently in Uspenskii [1955]. A related result also appeared in Nerode [1957]. Here we take $\theta \mapsto \widehat{\theta}$ to be an effective coding of the graphs of *finite* partial functions $\theta : \mathbb{N} \rightharpoonup \mathbb{N}$ as natural numbers.

Theorem 2.5 (Partial acting on partial). *A function* $F : \mathbb{N}^{\mathbb{N}}_{p\,rec} \rightharpoonup \mathbb{N}$ *is a partial effective operation iff*

- F *is monotone and continuous* (*i.e.*, F *preserves existing least upper bounds of chains in* $\mathbb{N}^{\mathbb{N}}_{p\,rec}$), *and*
- F *acts effectively on finite elements* (*i.e.*, *there is a partial recursive function* h *such that for every finite* $\theta : \mathbb{N} \rightharpoonup \mathbb{N}$ *we have* $F(\theta) = h(\widehat{\theta})$).

Moreover, every partial effective operation F *on* $\mathbb{N}^{\mathbb{N}}_{p\,rec}$ *extends uniquely to a monotone and continuous function* $\overline{F} : \mathbb{N}^{\mathbb{N}}_{p} \rightharpoonup \mathbb{N}$.

The other main positive result, often called the Kreisel-Lacombe-Shoenfield theorem, was obtained slightly later in Kreisel, Lacombe, and Shoenfield [1957], [1959], and independently in Tseitin [1959]. Our formulation here is somewhat less general than the original version.

Theorem 2.6 (Total acting on total). *A function* $F : \mathbb{N}^{\mathbb{N}}_{rec} \to \mathbb{N}$ *is a total effective operation iff it is the restriction of a uniformly partial recursive function* $\overline{F} : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}$ (*see Definition* 2.2) *whose domain contains* $\mathbb{N}^{\mathbb{N}}_{rec}$.

It follows that every total effective operation on $\mathbb{N}^{\mathbb{N}}_{rec}$ is continuous with respect to the usual Baire topology. Proofs of Theorem 2.6 may be found in many texts, *e.g.*, Rogers [1967], Beeson [1985], Aberth [1980], Odifreddi [1989]. A very short proof using Kleene's second recursion theorem was given by Gandy (Gandy [1962]), though it is perhaps too serendipitous for most ordinary mortals. A unified result subsuming both Theorems 2.5 and 2.6 may be found in Spreen and Young [1983].

Theorems 2.5 and 2.6 both provide very striking examples of the connection between computability and continuity. These results obviously say something deeper than the result of Grzegorczyk mentioned at the end of Section 2.3, since there we could interact with a type 1 function only as a black box, while here we have access to a recursive index for it. However, the final theorem of our quartet, due to Friedberg (Friedberg [1958b]), shows that the connection with continuity breaks down if totality and partiality are mixed:

THEOREM 2.7 (Partial acting on total). *There exists a partial effective operation on* $\mathbb{N}^{\mathbb{N}}_{rec}$ *which is not continuous* (*hence not the restriction of a Kleene partial recursive functional on* $\mathbb{N}^{\mathbb{N}}$).

The above theorems and their proofs are covered in Rogers [1967, §15.3].

Most of the remaining natural questions about type 2 computability turn out to have negative answers. For example, not every total effective operation on $\mathbb{N}^{\mathbb{N}}_{rec}$ is the restriction of a uniformly total recursive function on $\mathbb{N}^{\mathbb{N}}$. This can be shown using the celebrated *Kleene tree*, one of the most important counterexamples in the subject.

THEOREM 2.8 (Kleene tree). *There exists a binary tree B* (*i.e., a prefix-closed set of finite sequences over* $\{0, 1\}$) *such that*

- *B is primitive recursive* (*that is, there is a primitive recursive function b such that* $b\langle\alpha\rangle = 0$ *iff* $\alpha \in B$).
- *B contains finite paths of arbitrary length* (*so classically, by König's Lemma, B contains infinite paths*).
- *B contains no* recursive *infinite paths.*

The Kleene tree is so named after its appearance as Theorem LII of Kleene [1959b], although examples of the same phenomenon also appeared in Lacombe [1955d] and Zaslavskii [1955], Zaslavskii and Tseitin [1962]. We may now obtain a total effective operation Z that does not extend to a uniformly total recursive function on $\mathbb{N}^{\mathbb{N}}$, by defining

$$Z(f) = \mu n.\, [f(0), \ldots, f(n-1)] \notin B,$$

where μ is the minimization operator of ordinary recursion theory. This example also sheds light on the meaning of continuity in the foregoing results: Z is continuous on $\mathbb{N}^{\mathbb{N}}_{rec}$ by Theorem 2.6, but it cannot be extended even to a continuous function on the whole of $\mathbb{N}^{\mathbb{N}}$.

Two more negative results will help to complete the picture at type 2. Firstly, a plausible analogue of Theorem 2.5 fails for total functionals acting on total functions: not every continuous functional which acts effectively on a suitable basis of "finite" type 1 functions (namely, the eventually constant functions) is an effective operation. A simple counterexample was given in Pour-El [1960]; see also Rogers [1967, §15.3]. Secondly, the analogue of Theorem 2.6 for partial functionals acting on partial functions fails because the uniformly partial recursive functions $\mathbb{N}_p^{\mathbb{N}} \rightharpoonup \mathbb{N}$ are all *sequentially* computable, whereas the partial effective operations include *parallel* functions. This distinction was first made explicit in Platek [1966], and will be discussed at length in Section 4.

These early results show that the situation at type 2 is already quite complicated, and they serve to illustrate many of the general characteristics of the subject. Firstly, they exhibit a considerable diversity of approaches to defining a notion of "computable functional". Secondly, they show how quite different approaches sometimes lead to the same class of functionals, providing evidence for the intrinsic importance of this class (Theorem 2.6 is a good example of this). On the other hand, we have seen that not all definitions of computability conveniently collapse to a single notion, and that an important role is played by negative results and counterexamples. Finally, we have already seen several instances of the connection between computability and continuity.

§3. Total computable functionals.

Once various notions of type 2 computability had been considered, the possibility of trying to extend them to higher types was obvious. (Péter speculated informally on this possibility in Péter [1951b], but did not develop it.) The years 1957–59 saw the appearance of no fewer than four important notions of higher type computability, represented by Gödel's System T, Kleene's schemata S1–S9, the Kleene-Kreisel countable or continuous functionals (and their effective substructure), and Kreisel's hereditarily effective operations.

It is worth noting that all these notions were concerned primarily with hereditarily *total* functionals — good notions of computability for hereditarily partial functionals of all higher types were a later development (see Section 4). A qualification is needed here in the case of S1–S9, since (as we shall see) this notion naturally gives rise to partial computable functionals, albeit acting on hereditarily total objects. However, the partial functionals that arise in this way are somehow second-class citizens, since they cannot themselves serve as arguments to functionals of higher types. It therefore seems natural to include S1–S9 in our discussion of hereditarily total notions of computability.

Roughly speaking, each of the four notions mentioned above gave rise to its own strand of research, so that the study of higher type computability appears somewhat fragmented from 1960 onwards. We now consider these notions in turn and the lines of research they gave rise to.

3.1. System T and related type systems.

Gödel's System T, introduced in Gödel [1958], provides a higher type analogue of the notion of primitive recursive computation. It is one of a number of syntactic formalisms that define restricted classes of computable functionals, in the sense that the computational complexity of definable functionals is somehow limited. These formalisms fall somewhat outside our primary area of interest, since they make no claim to defining a "complete" class of computable functionals in any interesting sense, but they are close enough to our concerns to merit some attention.

3.1.1. *Gödel's T.* System T is essentially a simply typed λ-calculus over the ground type $\overline{0}$, with constants for zero, successor and primitive recursors of higher type:

$$\mathtt{0}\colon \overline{0}, \quad \mathtt{succ}\colon \overline{0} \to \overline{0}, \quad \mathtt{rec}_\sigma\colon \sigma \to (\sigma \to \overline{0} \to \sigma) \to \overline{0} \to \sigma.$$

These are equipped with the following reduction rules in addition to the usual β-rule and congruence rules given in Section 1.5.2:

$$\begin{aligned} \mathtt{rec}_\sigma UV\mathtt{0} &\rightsquigarrow U \\ \mathtt{rec}_\sigma UV(\mathtt{succ}\, n) &\rightsquigarrow V(\mathtt{rec}_\sigma UVn)n \end{aligned}$$

(where n is a variable of type $\overline{0}$). It is fairly easy to see that System T naturally extends Péter's notion of second order primitive recursion to higher types (see Section 2.3).

One can consider this system either as a standalone syntactic formalism, or as a language for denoting functionals in some given type structure, as in Section 1.5.2. For instance, if S is the full type structure over $\mathbb{N}$ defined in Section 1.5.1, then any closed term $U\colon \sigma$ of System T denotes an element $[\![U]\!] \in \mathsf{S}_\sigma$ in an obvious way.

In fact, Gödel's original presentation of System T was as a logical system for reasoning about higher type objects — in place of our reduction rules $U \rightsquigarrow U'$ he gave equational axioms $U = U'$. Gödel's purpose was to give an interpretation of first order Heyting arithmetic (the so-called *Dialectica interpretation*) in which first order sentences involving complex nestings of quantifiers were replaced by logically simple sentences involving objects of higher type. Specifically, Gödel gave a translation from formulae $\phi(\boldsymbol{x})$ of Heyting arithmetic to formulae $\phi'(\boldsymbol{x})$ of the form $\exists \boldsymbol{y}.\forall \boldsymbol{z}.\phi^*(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$, in which the variables in $\boldsymbol{y}, \boldsymbol{z}$ may be of arbitrary finite type, and ϕ^* is just a propositional combination of equations between terms of System T. For the sake of completeness we give the definition of the translation here, though the details are not too important for our purposes:

Definition 3.1 (Dialectica translation). *For atomic formulae α, define $\alpha' \equiv \alpha$. Given $\phi'(\boldsymbol{x}) \equiv \exists \boldsymbol{y}.\forall \boldsymbol{z}.\phi^*(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ and $\psi'(\boldsymbol{u}) \equiv \exists \boldsymbol{v}.\forall \boldsymbol{w}.\psi^*(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$, define*

$$\begin{aligned} (\phi \wedge \psi)' &\equiv \exists \boldsymbol{y}\boldsymbol{v}.\forall \boldsymbol{z}\boldsymbol{w}.\phi^*(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \wedge \psi^*(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) \\ (\phi \vee \psi)' &\equiv \exists \boldsymbol{y}\boldsymbol{v}t.\forall \boldsymbol{z}\boldsymbol{w}.(t = 0 \wedge \phi^*(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})) \vee (t = 1 \wedge \psi^*(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})) \end{aligned}$$

$$(\phi \Rightarrow \psi)' \equiv \exists \boldsymbol{V}\boldsymbol{Z}.\forall \boldsymbol{y}\boldsymbol{w}.\phi^*(\boldsymbol{x},\boldsymbol{y},\boldsymbol{Z}(\boldsymbol{y},\boldsymbol{w})) \Rightarrow \psi^*(\boldsymbol{u},\boldsymbol{V}(\boldsymbol{y}),\boldsymbol{w})^3$$
$$(\forall s.\phi)' \equiv \exists \boldsymbol{Y}.\forall s\boldsymbol{z}.\phi^*(\boldsymbol{x},\boldsymbol{Y}(s),\boldsymbol{z})$$
$$(\exists s.\phi)' \equiv \exists s\boldsymbol{y}.\forall \boldsymbol{z}.\phi^*(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$$

It can be shown that if ϕ is provable in HA then there are terms $\boldsymbol{Y}$ of System T such that $\phi^*(\boldsymbol{x},\boldsymbol{Y}(\boldsymbol{x}),\boldsymbol{z})$ is provable in System T using just *quantifier-free* intuitionistic logic. The terms $\boldsymbol{Y}$ here can be thought of as embodying the constructive content of the proof of ϕ. We may now interpret the terms $\boldsymbol{Y}$, and the proof of $\phi^*(\boldsymbol{x},\boldsymbol{Y}(\boldsymbol{x}),\boldsymbol{z})$, in any type structure satisfying the axioms of System T. We thus obtain a "functional interpretation" of Heyting arithmetic, which can be used to prove the consistency of Heyting arithmetic, and hence (in view of Gödel's double-negation translation) that of Peano arithmetic.[4]

However, the interest of System T seems to go beyond this original application. Gödel himself gave a proof-theoretic characterization of the expressive power of System T: the T-definable type 1 functions (in S, say) are precisely the functions provably total in first order Heyting or Peano arithmetic. Grzegorczyk (Grzegorczyk [1964]) gave some alternative characterizations of the System T definable functionals in the style of combinatory logic. Tait and several others (Tait [1967], Dragalin [1968], Hinata [1967], Hinatani [1966], Sanchis [1967], Shoenfield [1967]) proved independently that System T is *strongly normalizing* — that is, for any System T term, all reduction paths terminate yielding the same final value, or *normal form*. (A remarkable alternative proof of this was later given in Gandy [1980].) This important result implies that any term model for System T is itself a type structure with all the properties needed for the Dialectica interpretation. Thus, a functional interpretation of arithmetic can be given in terms of System T itself, without the need for an interpretation of System T in another type structure. (This idea was already implicit in a remark in Kreisel [1959, §3.4].)

The Dialectica interpretation also inspired work on interpretations of systems for analysis (or second order arithmetic) in a similar spirit (Kreisel [1959], Spector [1962]); however, these go beyond System T itself. For instance, Spector's system includes an additional operator for defining objects by means of

[3]Here, for instance, if $\boldsymbol{Z}$ abbreviates $Z_1 \dots Z_r$, we write $\boldsymbol{Z}(\boldsymbol{y},\boldsymbol{w})$ to abbreviate $Z_1(\boldsymbol{y},\boldsymbol{w})$, $\dots, Z_r(\boldsymbol{y},\boldsymbol{w})$. Gödel motivated the clause for implication as follows. We identify a proposition $(\exists \boldsymbol{y}.\cdots) \Rightarrow (\exists \boldsymbol{v}.\cdots)$ with the existence of computable functions $\boldsymbol{V}$ that to each sequence $\boldsymbol{y}$ making the antecedent true assign a sequence $\boldsymbol{v}$ making the consequent true. Moreover, we identify a proposition $(\forall \boldsymbol{z}.\cdots) \Rightarrow (\forall \boldsymbol{w}.\cdots)$ with the existence of computable functions $\boldsymbol{Z}$ that to each sequence $\boldsymbol{w}$ making the consequent false assign a sequence $\boldsymbol{z}$ making the antecedent false.

[4]All that is required for Gödel's argument is that *some* type structure satisfying the axioms of System T exists. In Gödel [1958], Gödel seemingly regarded the existence of a suitable notion of computable operation of finite type as immediately apparent. A footnote in a revised version of the paper (Gödel [1972]), however, suggests he had in mind the structure HRO (see Section 3.4).

bar recursion (an intuitionistic principle considered by Brouwer); this system admits a natural interpretation in the type structure of total continuous functionals (see Section 3.3).

Another set of questions concern *theories* for System T. For example, what equivalence relations on System T terms are induced by their interpretation in various type structures? The largest reasonable theory, which we shall call $\approx$, is the one given by the notion of "observational equivalence": $M \approx N$ at type σ iff for all closed $P\colon \sigma \to \overline{0}$, PM and PN reduce to the same normal form. An argument due to Kreisel (appearing as one of the final set of exercises in Barendregt [1984]) shows that the interpretation of System T in S induces a strictly smaller theory — in computer science terms, this interpretation is not *fully abstract*. By contrast, Loader has shown that the interpretation of System T in the type structure HEO (see Section 3.4) induces exactly the theory $\approx$ (Loader [1997]).

Other mathematical results pertaining specifically to System T include the theorems of Howard (Howard [1973]; see also Girard [1987, annex 7.A]) that all T-definable functionals are *hereditarily majorizable*, and hence that there is no T-definable *modulus of extensionality* functional. A further mathematical analysis of System T in terms of functors over the category of ordinals is given in Päppinghaus [1985] (see also Girard [1988]).

Further discussions of System T can be found in Kreisel [1959], Troelstra [1973, §3.5], Barendregt [1984, Appendix A.2], and Girard [1987, annex 7.A]. A good survey of later research related to System T can be found in Section 5 of Troelstra's introductory note to Gödel's original paper in Gödel [1990]. For more on the Dialectica interpretation, see Avigad and Feferman [1998].

3.1.2. *Kleene's S1–S8.* We have already observed that System T can be interpreted in various type structures, such as the full set-theoretic type structure S. Indeed, one reason why System T is interesting from our point of view is that it admits an interpretation of this kind in practically all the type structures we shall have occasion to consider. It therefore provides a kind of common "skeleton" for all these type structures, and gives us an effectively enumerable class of "total" functions that can play a role analogous to that of the primitive recursive functions in ordinary recursion theory.

However, in this regard there is nothing particularly unique about System T — many other systems would serve the same purpose. For instance, a weaker form of primitive recursion is given by the typed λ-calculus with constants 0, succ and

$$\widehat{\mathtt{rec}}_\sigma \colon \sigma \to (\sigma \to \overline{0} \to \sigma) \to \overline{0} \to \sigma$$

together with reduction rules

$$\widehat{\mathtt{rec}}_\sigma UV0\boldsymbol{W} \rightsquigarrow U\boldsymbol{W}$$

$$\widehat{\mathtt{rec}}_\sigma UV(\mathtt{succ}\ n)\boldsymbol{W} \rightsquigarrow V(\widehat{\mathtt{rec}}_\sigma UVn)n\boldsymbol{W}$$

where $\sigma = \sigma_1 \to \cdots \to \sigma_n \to \overline{0}$ and $\boldsymbol{W}$ textually abbreviates $W_1 \dots W_n$.[5] This system, which we will call System $\widehat{\mathrm{T}}$, corresponds exactly in expressive power to Kleene's schemata S1–S8 (see Section 3.2 below); Kleene referred to the elements of S definable by these means as the *primitive recursive functionals*. Whereas System T offers a natural higher type generalization of Péter's notion of type 2 primitive recursiveness, System $\widehat{\mathrm{T}}$ offers a similar generalization of Kleene's notion (Definition 2.2(i)). Thus, System T can also define functions such as the Ackermann function, whereas the type 1 functions definable in System $\widehat{\mathrm{T}}$ are just the primitive recursive functions in the usual sense (see Kleene [1959b, §1]).

In some sense S1–S8 offers a better common core of "simple" functionals than System T, insofar as it can be interpreted in a wider class of type structures. It would seem, though, that overall Kleene's S1–S8 holds less mathematical interest than Gödel's T.

System T also provided a point of departure for two other strands of later work, namely modern type theory (which typically considers stronger systems than System T), and higher type complexity theory (which typically considers weaker systems). Both of these are significant areas of research in theoretical computer science. For detailed information on modern type theory, a good reference is Barendregt [1992]; for higher type complexity, we refer the reader to the surveys in the recent articles of Irwin, Kapron, and Royer [2001a], [2001b].

3.2. Kleene computability: S1–S9.

3.2.1. *Kleene's work.* The first serious attempt at a full-blown generalization of the notion of recursive function to all type levels was Kleene's definition of higher type computability via the schemata S1–S9 (Kleene [1959b], [1963]). This can be seen as generalizing his notion of partial recursiveness at type 2 (Definition 2.2). In Kleene's approach to higher type computation, we suppose we are given a type structure A of hereditarily total functionals over $\mathbb{N}$, and we define a class of computable *partial* functionals over A by means of certain computation schemata. However, as mentioned at the beginning of Section 3, the computable partial functionals appear to play a less central role than the total ones.

The spirit behind Kleene's definition is easily grasped: we are allowed to perform effective computations involving elements of A, in which the elements of function type are treated simply as *oracles* or *black boxes*. In particular, we may feed an element of function type with numbers or functions that are themselves computable (in the same sense), and observe the numerical results, but we are not granted access to information about such elements in any other way. Thus, Kleene's definition embodies the ideal of computing with

[5]The notation $\widehat{\mathtt{rec}}$ is adapted from Feferman [1977b], where the constants $\mathtt{rec}_\sigma$ are called $\mathbb{N}$-*recursion operators* and the $\widehat{\mathtt{rec}}_\sigma$ are called *elementary recursion operators*.

functions as pure extensions, without reference to how they are represented or implemented. Indeed, A will in general include non-computable objects, so that we have to imagine the corresponding oracles as working "by magic".

Kleene in fact concentrated his attention on computations over objects of *pure* type (see Section 1.5.1), though this was not an especially significant decision. Because of its historical importance, we reproduce Kleene's original definition here in all its glory, with slightly modified notation.[6] Given a type structure A, we write $X(A)$ for the disjoint union of all sets $A_{\sigma_1} \times \cdots \times A_{\sigma_r}$ $(r \geq 1)$ where the σ_i are pure types. We write $\#-$ for the coding of pure types as natural numbers given by $\#\overline{n} = n$, and take $\langle \cdots \rangle$ to be an effective coding of finite sequences of natural numbers. Throughout this section we will use $x : \sigma$ as an abbreviation for $x \in A_\sigma$, rather than as a formal expression. We also abbreviate $x_1, \ldots, x_r$ by $\boldsymbol{x}$, $x_1 : \sigma_1, \ldots, x_r : \sigma_r$ by $\boldsymbol{x} : \boldsymbol{\sigma}$, and $\langle \#\sigma_1, \ldots, \#\sigma_r \rangle$ by $\#\boldsymbol{\sigma}$.

The definition proceeds by introducing a system of *indexing* whereby natural numbers m encode definitions of computable partial functions $\{m\}$; this is reminiscent of the indexing of partial recursive functions in ordinary recursion theory.

Definition 3.2 (Kleene computability). *Let A be an extensional type structure over $\mathbb{N}$.*

(i) *We inductively define a partial function $\{-\}(-) : \mathbb{N} \times X(A) \rightharpoonup \mathbb{N}$, called* index application, *by means of the following clauses (which we interpret as applying whenever they are well-typed).*

S1: Successor function: $\{\langle 1, \#\boldsymbol{\sigma} \rangle\}(\boldsymbol{x} : \boldsymbol{\sigma}) = x_1 + 1$.

S2: Constant functions: *For any $q \in \mathbb{N}$*, $\{\langle 2, \#\boldsymbol{\sigma}, q \rangle\}(\boldsymbol{x} : \boldsymbol{\sigma}) = q$.

S3: Projection: $\{\langle 3, \#\boldsymbol{\sigma} \rangle\}(\boldsymbol{x} : \boldsymbol{\sigma}) = x_1$.

S4: Composition: *For any $g, h \in \mathbb{N}$,*

$$\{\langle 4, \#\boldsymbol{\sigma}, g, h \rangle\}(\boldsymbol{x} : \boldsymbol{\sigma}) \simeq \{g\}(\{h\}(\boldsymbol{x}), \boldsymbol{x}).$$

S5: Primitive recursion: *For any $g, h \in \mathbb{N}$, if $m = \langle 5, \#\boldsymbol{\sigma}, g, h \rangle$ then*

$$\{m\}(0, \boldsymbol{x}) \simeq \{g\}(\boldsymbol{x}), \quad \{m\}(n+1, \boldsymbol{x}) \simeq \{h\}(n, \{m\}(n, \boldsymbol{x}), \boldsymbol{x}).$$

S6: Permutation of arguments: *For any $g \in \mathbb{N}$ and $1 \leq k < r$,*

$$\{\langle 6, \#\boldsymbol{\sigma}, k, g \rangle\}(\boldsymbol{x} : \boldsymbol{\sigma}) \simeq \{g\}(x_{k+1}, x_1, \ldots, x_k, x_{k+2}, \ldots, x_r).$$

S7: Type 1 application: *If $\sigma_1 = \overline{1}$ and $\sigma_2 = \overline{0}$, then*

$$\{\langle 7, \#\boldsymbol{\sigma} \rangle\}(\boldsymbol{x} : \boldsymbol{\sigma}) = x_1(x_2).$$

[6]Kleene's definition was restricted to the case $A = \mathsf{S}$. Moreover, Kleene adopted the peculiar convention that only the order of arguments within each type was material, so his version of S6 was slightly different from the one given here.

S8: Higher type application: *For any $h \in \mathbb{N}$ and $t \geq 2$, we have*

$$\{\langle 8, \#\boldsymbol{\sigma}, t, h\rangle\}(y : \overline{t}, \boldsymbol{x} : \boldsymbol{\sigma}) \simeq y(\Lambda z : \overline{t-2}.\, \{h\}(y, z, \boldsymbol{x}))$$

provided $\{h\}(y, z, \boldsymbol{x})$ is defined for all $z \in A_{t-2}$.

S9: Index invocation: $\{\langle 9, \#\boldsymbol{\sigma}, \#\boldsymbol{\tau}\rangle\}(n : \overline{0}, \boldsymbol{y} : \boldsymbol{\sigma}, \boldsymbol{z} : \boldsymbol{\tau}) \simeq \{n\}(\boldsymbol{y})$.

(ii) *We say $F : A_{\sigma_1} \times \cdots \times A_{\sigma_r} \rightharpoonup \mathbb{N}$ is* Kleene computable over *A if there is a number m (called an* index *for F) such that $F(\boldsymbol{x}) \simeq \{m\}(\boldsymbol{x})$ for all $\boldsymbol{x}$.*

One may wonder why Kleene, one of the pioneers of the λ-calculus, did not make greater use of the typed λ-calculus in formulating this notion of computability. It seems that his early attempts to give talks based on his λ-calculus work before the latter had achieved wide currency were less well-received than the above definition via indices. A definition of Kleene computability in a more modern spirit will be given in Part II.

As already noted, S1–S8 by themselves define a perfectly good class of *total* functionals — the schema S9 is the only one that introduces partiality into the definition. Notice that even if the indexing system were redesigned so that every natural number indexed precisely one function of each type, partial functions would still arise for reasons of circularity. Specifically, if m were an index for the function $F = \Lambda n.\{n\}(n)$ then $F(m)$ would be undefined according to the inductive definition above, since the clause for S9 would tell us merely that $\{m\}(m) \simeq \{m\}(m)$. That is, a value for $\{m\}(m)$ would appear at some stage in the inductive generation of $\{-\}(-)$ only if it had already appeared at some previous stage.

Kleene generally conceived computations in this setting as infinitely branching trees of transfinite depth — when invoking S8 one computes the entire graph of $\Lambda z.\{h\}(y, z, \boldsymbol{x})$ by means of auxiliary computations before presenting it to the oracle y. This is perhaps not the only possible conception: for example, one might imagine that the magic of the oracles included the ability to recognize functions from finitary descriptions of them, in which case these auxiliary computations would not be needed. But whatever conception one adopts, Kleene's schemata exhibit a curious tension between effective, finitary computational processes and calls to numinous or infinitistic oracles, so that the computational significance of Kleene's definition is rather difficult to gauge.

Kleene's motivation seems to have sprung from two distinct strands of his earlier work. On the one hand, he was certainly seeking an appropriate higher type generalization of the basic notions of ordinary recursion theory — including, if possible, some analogue of Church's thesis at higher types. On the other hand, he was seeking to explain his theory of logical complexity for classical predicate logic (embodied in the arithmetic and analytic hierarchies, as in Kleene [1955a], [1955b]) in terms of computability relative to certain higher type objects (hence the emphasis on *quantifiers* in his papers on higher type computability). This latter motivation goes some way towards explaining two features of Kleene's definition that sometimes appear puzzling to modern readers: the restriction to computations on *total* functionals, and the specialization to the type structure S rather than any more constructively given class of functionals.

Kleene developed some basic results for this notion of computability analogous to those of ordinary recursion theory: for example, higher type versions of the normal form theorem (Kleene [1959b, §5]), and a restricted version of the first recursion theorem (Kleene [1963, §10]). He also showed that the schema S9 is strictly more powerful than the μ-recursion (minimization) schema familiar from ordinary recursion theory (Kleene [1959b, §8]), and argued that the former gives rise to a more compelling notion of higher type computability. In addition, Kleene introduced the natural notion of relative computability in the setting of S1–S9:

Definition 3.3 (Relative Kleene computability).[7]

(i) *A partial function* $g : A_{\sigma_1} \times \cdots \times A_{\sigma_r} \rightharpoonup \mathbb{N}$ *is* Kleene computable relative to $y \in A$ *if there is a Kleene computable partial function* f *such that* $f(y, \mathbf{z}) \simeq g(\mathbf{z})$ *for all* $\mathbf{z} : \boldsymbol{\sigma}$.

(ii) *Given* $x, y \in A$ *of type level* > 0, *we write* $x \preceq y$ *if* x *is Kleene computable relative to* y. *We say* x, y *are of the same* Kleene degree *if* $x \preceq y \preceq x$.

Other results made precise the connection with logical complexity. For instance:

Theorem 3.4 (Kleene [1959b, §10]). *A total function* $F : \mathbb{N}^r \to \mathbb{N}$ *is hyperarithmetical (that is, its graph is definable by a* Δ^1_1 *formula) iff it is Kleene computable relative to the type* 2 *object* ${}^2\exists$, *given by*

$$
{}^2\exists(f) = \begin{cases} 0 & \textit{if } \exists n.\, f(n) = 0, \\ 1 & \textit{otherwise.} \end{cases}
$$

Kleene also followed up his definition via S1–S9 with a clutch of papers (Kleene [1962c], [1962d], [1962b], [1962a]) showing that several alternative definitions of higher type computability — via Turing machines, λ-calculus,[8] and Herbrand-Gödel style recursive definitions — give rise to the same class of computable functionals. These results closely parallel the corresponding results of ordinary recursion theory, but they are not especially deep extensions of the familiar results, and all these definitions are based on essentially the same idea of computation with oracles. Nevertheless, these results go some way towards establishing the robustness of Kleene's class of computable functionals.

[7] Kleene's original definition did not take exactly this form, but the equivalence is trivial.

[8] In fact, in Kleene [1962b] Kleene used a system based on what is now known as Church's λI calculus.

Despite these reassuring results, however, Kleene's notion presents us with some strange anomalies. We mention two of these here, both arising somehow from the curious infinitary side-condition in S8.

The first of these is the notorious fact that Kleene's computable partial functionals are not closed under some basic substitution operations. For example, consider the functions h, Φ, G defined by

$$\begin{aligned} h(m\colon \overline{0}, n\colon \overline{0}) &\simeq \begin{cases} 0 & \text{if } \neg T(m,m,n), \\ \text{undefined} & \text{if } T(m,m,n) \end{cases} \\ \Phi(F: \overline{2}, m\colon \overline{0}) &\simeq F(\Lambda n.h(m,n)) \\ G(f: \overline{1}) &= 0 \end{aligned}$$

where T is the T-predicate of ordinary recursion theory. Then the partial functions Φ, G are both Kleene computable, but $\Lambda e.\Phi(G,e)$ is not. This is because $G(\Lambda n.h(m,n)) = 0$ iff $\Lambda n.h(m,n)$ is a total function, which is the case iff $\phi_m(m)$ is *not* defined. Hence, if $\Lambda e.\Phi(G,e)$ were computable, we could solve the halting problem.

This is obviously rather worrying — indeed, it even seems questionable in what sense we have defined a computable functional Φ if we are not even allowed to plug in the total computable functional G as its first argument. Essentially the same problem lies behind the failure of the natural generalization of the first recursion theorem. There is also other evidence that the notion of partial computable functional here is pathological: for instance, a set that is semi-recursive and co-semi-recursive need not be recursive (see Platek [1966, p. 131]), and the union of two semi-recursive sets need not be semi-recursive (see Grilliot [1969b, p. 233]).

One way of responding to this problem is to restrict one's attention to the *total* computable functionals. In most interesting cases, the total Kleene computable functions over A constitute a well-behaved class, so we may regard Kleene's definition as picking out an important substructure of A:

Definition 3.5. *A type structure A is* closed under Kleene computation *if all the total Kleene computable functionals over A are themselves elements of A. We then write* $\mathsf{KC}(A)$ *for the type structure consisting of the total Kleene computable elements of A.*

In this situation, the elements of $\mathsf{KC}(A)$ behave well with respect to substitution and indeed constitute a cartesian closed category.

A second curious feature of Kleene's definition, less often noted, is its apparent *non-absoluteness*. The set of indices that define total functionals, for instance, may vary from one type structure A to another — intuitively, the fewer elements of type σ there are in A, the easier it is for an index to define a total functional of type $\sigma \to \overline{0}$. Even for the case of computability over S, it is unclear *a priori* whether the totality or otherwise of the function $\{m\}(-)$

is an absolute property of m, unless one is willing to accept the notion of "the" full set-theoretic type structure as absolute. Could there be Kleene-style computations whose termination is dependent on controversial principles of set theory? We will return to this question in Part II.

Other presentations of the Kleene computable functionals were later given in Gandy [1967a] and Platek [1966]. Both of these treatments sought to avoid some of the arbitrary features of Kleene's definitions and to emphasize the naturalness of this notion of computability. Gandy gave a more perspicuous definition involving register machines, and argued that one was led ineluctably to Kleene's notion of higher type computability if (a) one restricted attention to (hereditarily) total arguments, and (b) one treated functions as "pure extensions". Platek gave a characterization of Kleene's functionals within a general framework for recursion theory that stressed *definability* of functions by recursion rather than computability. In particular, he showed how a very natural theory of recursive definability could be developed for type structures of hereditarily *partial* functionals — here the troublesome side-condition in S8 is not needed, and one may easily recover Kleene's original notion via some simple relationships between partial and total type structures. (We will say more about this approach in Section 4.1.)

Platek's approach via inductive definability was streamlined by Moschovakis (Moschovakis [1976]), whose approach made use of ideas from his work on abstract computability theories (Moschovakis [1969], [1974a]), which in turn was inspired by Kleene's original definition of S1–S9. A key insight here was that higher type computability can in some sense be reduced to type 2 computability: if we think of each of the sets A_σ as a separate ground sort, then Kleene computability over A becomes simply a matter of what can in an abstract sense be computed relative to the (first order) application functions and the (second order) λ-abstraction operations. This treatment was followed in the expository article of Kechris and Moschovakis [1977]. A comparison of various approaches to Kleene computability was given in Fenstad [1978] (see also Feferman [1977a]).

Beyond the results already mentioned, however, the pure notion of Kleene computability over S appears to hold rather little mathematical interest. For this reason, the later study of recursion theory on S concentrated almost entirely on questions of relative computability or on certain kinds of non-computable object. Thus, at this point the subject largely lost contact with genuinely effective notions of computability. Nevertheless, the ideas involved can still be seen as plausible generalizations of "computation" to infinite objects, so an account of these developments is in order.

3.2.2. *Recursion in normal objects.* Most of the later work concentrated on the theory of *normal* functionals in S:

DEFINITION 3.6 (Normal functionals).

(i) *We write* ${}^{k}\exists$ $(k \geq 2)$ *for the object of* S_k *embodying existential quantification over* S_{k-2}.
(ii) *A functional* Θ *of type level* k *is said to be* normal *if* ${}^{k}\exists \preceq \Theta$.

We will be interested in notions of Θ-*computability* (that is, Kleene computability relative to Θ) where Θ is normal, and particularly in the notions of ${}^{k}\exists$-*computability*.

The study of normal objects and the associated notions of computability turns out to yield a very rich and beautiful theory. Roughly speaking, if ${}^{k}\exists$ is deemed computable, then the theory of computability is good up to and including type level k. One way to see intuitively why this should be so is to note that in a setting where we have the ability to quantify over S_{k-2}, it is consonant to think of the infinitary side-condition in S8 as somehow "semidecidable" (and hence harmless) for $t \leq k$; thus, the anomalies mentioned earlier for pure Kleene computability are washed out in this setting.

The significance of normal objects had already emerged from Kleene's early work, but their good properties were first seriously exploited by Gandy (Gandy [1967b]),[9] who introduced the key concepts of stage comparison and number selection. Informally, a *stage comparison function* for two functionals F and G is a partial function $\chi_{F,G}(\boldsymbol{x}, \boldsymbol{y})$ that tells us which out of the Kleene computation trees for $F(\boldsymbol{x})$, $G(\boldsymbol{y})$ has the smaller ordinal depth, assuming that at least one of these trees is well-founded. In particular, it can tell us which of the two computations will terminate, given that at least one of them will. The following theorem was stated for type 2 in Gandy [1967b], and extended to higher types (independently, and by different methods) in Platek [1966] and Moschovakis [1967], [1976]:

THEOREM 3.7 (Stage comparison). *Let* Θ *be a normal functional of level* k. *If* F, G *are* Θ-*computable and of level* $\leq k$, *then* $\chi_{F,G}$ *is also* Θ-*computable.*

One can often use stage comparison where in ordinary recursion theory one might use interleaving arguments — for instance, in showing that (for Θ normal of level k) the union of two Θ-semidecidable subsets of S_k is Θ-semidecidable, or that a subset of S_k is Θ-decidable iff both it and its complement are Θ-semidecidable.

Given a predicate $P(\boldsymbol{x}, n : \overline{0})$, a *selection function* for P is a partial functional $F(\boldsymbol{x})$ that "selects" a value of n satisfying $P(\boldsymbol{x}, n)$ whenever there is one. Again, the basic result is proved for type 2 in Gandy [1967b], and generalized in Moschovakis [1967], [1976], Platek [1966]:

THEOREM 3.8 (Number selection). *Let* Θ *be normal of level* $k \geq 2$. *For any* Θ-*semidecidable predicate* $P(\boldsymbol{x}, n : \overline{0})$ *on* S_k, *there is a* Θ-*computable partial*

[9]Gandy first presented these results around 1962.

function $F(\boldsymbol{x})$ such that

$$\forall \boldsymbol{x}.\, ((\exists n.\, P(\boldsymbol{x}, n)) \Longrightarrow F(\boldsymbol{x})\downarrow \wedge P(\boldsymbol{x}, F(\boldsymbol{x}))).$$

It follows, for example, that the class of Θ-semidecidable subsets of S_k is closed under (uniform) $\mathbb{N}$-indexed unions, and that a partial function of level k is Θ-computable iff its graph is Θ-semidecidable. The basics of stage comparison and number selection up to this point are covered in Kechris and Moschovakis [1977].

A selection theorem of another kind is the following. It first appeared in Grilliot [1969b] but with an incorrect proof. A correct proof was given in Harrington and MacQueen [1976].

Theorem 3.9 (Grilliot selection). *Let Θ be normal of level $k + 2$ ($k \geq 1$). For any inhabited Θ-semidecidable subset P of S_{k-1}, there is an inhabited Θ-decidable subset Q such that $Q \subseteq P$.*

Much attention has also been devoted to the study of *sections* as defined in Kleene [1963]. Given any $x \in \mathsf{S}$, the *section* of x is the set $\{y \in \mathsf{S} \mid y \preceq x\}$; likewise the *$k$-section* of x is $\{y \in \mathsf{S}_k \mid y \preceq x\}$.[10] One of the high points of the theory is the following result due to Sacks (Sacks [1974], [1977]):

Theorem 3.10 (Plus-one theorem). *Suppose $0 < k < n$. Then for any normal object $\Theta \in \mathsf{S}_n$ there is a normal object $\Psi \in \mathsf{S}_{k+1}$ such that Θ and Ψ have the same k-section.*

The proof of this very substantial theorem exploits some beautiful connections with admissible set theory (Sacks [1971]). Let us consider the case $k = 1$ as an example. Observe that any hereditarily countable set X (that is, a set whose transitive closure is countable) can be coded up as a type 1 function: essentially, one just needs to code up the binary membership relation on the transitive closure of X. Under this correspondence, those sets of type 1 functions which arise as sections correspond precisely to those sets of hereditarily countable sets which constitute models for a certain weak fragment of set theory. Sacks was able to apply forcing techniques to such models in order to construct a suitable type 2 object Ψ for the above theorem.

Sacks also considered higher type analogues of problems from classical degree theory such as Post's problem (Sacks [1980], MacQueen [1972]; see also Section 3.2.4 below).

Sections are to total functionals what *envelopes* are to partial functionals. If $\sigma = \sigma_1 \to \cdots \to \sigma_r \to \overline{0}$, let us write $\mathsf{S}_{p\sigma}$ for the set of all partial functions $\mathsf{S}_{\sigma_1} \times \cdots \times \mathsf{S}_{\sigma_n} \rightharpoonup \mathbb{N}$. The *$k$-envelope* of $x \in \mathsf{S}$ is the set

$$\{f \in \mathsf{S}_{p\sigma} \mid \mathit{level}\,(\sigma) \leq k \wedge f \text{ Kleene computable relative to } x\}$$

[10]The k-section is sometimes taken to include all objects of type level $\leq k$ computable in x, but it makes little difference.

Moschovakis (Moschovakis [1974c]) showed that the analogue of the plus-one theorem for envelopes fails: indeed, if Θ is *any* normal type 3 object, the 1-envelope of Θ is not the 1-envelope of any normal type 2 object. However, we have the following result due to Harrington (Harrington [1973]):

THEOREM 3.11 (Plus-two theorem). *Suppose* $0 < k < n - 1$. *Then for any normal object* $\Theta \in \mathsf{S}_n$ *there is a normal object* $\Psi \in \mathsf{S}_{k+2}$ *such that* Θ *and* Ψ *have the same* k*-envelope.*

Regarding elements of the k-envelope of a normal Θ as representing Θ-semidecidable predicates, it is natural to ask what closure properties these enjoy. It is clear from earlier remarks that the 1-envelope of a normal type 2 object is closed under existential quantification over $\mathbb{N}$ (in an obvious sense). However, the situation at higher types is more interesting:

THEOREM 3.12. *Let* Θ *be normal of level* $m \geq 3$. *Then the* $(m-1)$*-envelope of* Θ *is closed under existential quantification over* S_j *for* $j < m - 2$ (*Harrington and MacQueen* [1976]), *but not under existential quantification over* S_{m-2} (*Moschovakis* [1967], *Grilliot* [1967]).

Both Sacks and Harrington worked with definitions of recursion in normal functionals that differed somewhat from Kleene's, but the equivalences are verified in Lowenthal [1976]. The proofs of many of the above results have been clarified by adopting the perspective of abstract recursion theory, which isolates the essential features of the domains (in this case, the S_σ) over which we are computing. This abstract approach is worked out in the books Moldestad [1977], Fenstad [1980].

3.2.3. *Hierarchies.* Another area of investigation has been the search for *hierarchies* for the functionals computable from a given object. The starting point for this endeavour is the fact that the 1-section of ${}^2\exists$ consists of exactly the hyperarithmetic or Δ^1_1 functions (Theorem 3.4). As shown in Kleene [1955b], these can be classified according to their logical complexity by means of an ordinal hierarchy of height ω_1^{CK}, known as the hyperarithmetic hierarchy. Kleene in Kleene [1963] introduced the general problem of seeking similar ordinal stratifications for the sections of other higher type objects.

Tugué (Tugué [1960]) considered in particular the normal type 2 object E_1 embodying the *Suslin quantifier*:

$$E_1(f : \overline{1}) = \begin{cases} 0 & \text{if } \exists g : \overline{1}.\, \forall n.\, f(\widetilde{g}(n)) = 0, \\ 1 & \text{otherwise,} \end{cases}$$

where $\widetilde{g}$ is as defined in Section 1.4. Tugué and later Richter (Richter [1967]) gave ordinal hierarchies for the 1-section of E_1. Kleene had conjectured that this class might coincide with the Δ^1_2 functions, but this was refuted by Shoenfield (Shoenfield [1962]).

Some of these hierarchies were considered in greater detail in Gandy [1967b], Grilliot [1969a]. In particular, Gandy showed how such hierarchies could be used to prove structural results such as the stage comparison and number selection theorems mentioned in Section 3.2.2.

A pleasing generalization of these hierarchy results to the 1-section of an arbitrary normal type 2 object was given independently by Shoenfield (Shoenfield [1968]) and Hinman (Hinman [1966], [1969]). The two versions are essentially the same; the key point in both cases is that the set of ordinal notations, rather than being fixed in advance, is generated simultaneously with the hierarchy itself. Later, Wainer (Wainer [1974]) showed how a somewhat more delicate construction yields a hierarchy for the 1-section even of an arbitrary non-normal type 2 object (see also Wainer [1975], [1978]).

A natural challenge was to obtain similar hierarchy results for 2-sections of ${}^3\exists$ and other type 3 objects. In Kleene [1963], Kleene outlined the construction of a *hyperanalytic hierarchy* for type 2 objects; this hierarchy was studied in detail by his student Clarke (Clarke [1964]), who conjectured that it does not exhaust the type 2 objects computable in ${}^3\exists$. The conjecture was confirmed by Moschovakis (Moschovakis [1967]), who however constructed an alternative hierarchy which does exhaust them, by using a much more powerful system for generating ordinal notations.

Other objects of interest are the *superjump* operators ${}^kS$ $(k \geq 2)$, defined by

$$ {}^kS(e : \overline{0}, x : \overline{k-1}) = \begin{cases} 0 & \text{if } \{e\}(x)\downarrow, \\ 1 & \text{if } \{e\}(x)\uparrow. \end{cases} $$

The operator ${}^2S$ is the ordinary jump operator of classical recursion theory, which is equivalent in strength to ${}^2\exists$. The superjump operator ${}^3S$, however, is strictly weaker than ${}^3\exists$ (and hence non-normal). Recursion in ${}^3S$ has been studied in Gandy [1967b], Platek [1971], Aczel and Hinman [1974]. Superjump operators of even higher types have been considered in Harrington [1973], [1974]. Much more recently, both ${}^3\exists$ and ${}^3S$ have been used in the construction of universes of domains closed under powerful type formation principles (Normann [1997], [1999b]).

Many of the above hierarchy results are covered in detail in Hinman [1978].

3.2.4. *Set recursion.* It was discovered by Normann (Normann [1978b]), and independently by Moschovakis, that the ideas of Kleene computability relative to all the ${}^k\exists$ could naturally be generalized to a theory of computability on arbitrary sets, in which the equality predicate on sets is deemed to be computable. This theory is called *E-recursion theory*; it has close connections with Gödel's notion of constructibility. (The connection had essentially been noted already in Sacks [1971].)

This much more general setting allowed the whole subject to break free from the shackles of the type structure S, and E-recursion was found to provide a natural framework for the later degree-theoretic investigations of Sacks and others (Sacks [1985], [1986], Griffor [1980], Slaman [1981], [1985]). Thus, at this point the study of higher type recursion became subsumed in admissible set theory, and specific references to the finite types are rare in the later papers.

For a full treatment of E-recursion theory see Sacks [1990]; for a survey see Sacks [1999].

3.3. The total continuous functionals.

3.3.1. *Early work: Kleene and Kreisel.* Whereas computations over the full set-theoretic type structure S have connections with the metamathematics of classical logic, the study of the *total continuous functionals* has largely been driven by interest in more constructive interpretations. The type structure C of total continuous functionals was obtained around the same time by Kleene (Kleene [1959a]) (who called them the *countable* functionals) and Kreisel (Kreisel [1959]) (who called them the *continuous* functionals). Both definitions were rather complicated and neither were *prima facie* very natural, but they both embodied the basic idea that any finite piece of information about the output from a functional F was determined by a finite amount of information about the input.

Kleene's definition centres around the observation that any total continuous type 2 function $F\colon \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ can be completely determined by a type 1 function — for instance, by the function $f_F\colon \mathbb{N} \to \mathbb{N}$ defined (classically) by

$$f_F\langle n_0, \ldots, n_{r-1}\rangle = \begin{cases} m+1 & \text{if } F(g) = m \text{ for all } g \text{ such that} \\ & \quad g(i) = n_i \text{ for all } i < r, \\ 0 & \text{if no } m \text{ exists with this property.} \end{cases}$$

Reversing this idea, we can define a partial "application" operation $- \mid -$: $\mathbb{N}^{\mathbb{N}} \times \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}$ by

$$\begin{aligned} f \mid g &\simeq f(\tilde{g}(r)) - 1, \\ &\quad \text{where } r \text{ is the least number such that } f(\tilde{g}(r)) > 0. \end{aligned}$$

Thus, type 1 objects can be used to encode type 2 objects. Once this is in place, it easy to see how type 1 objects can be used to encode objects of arbitrary pure types:

Definition 3.13 (Kleene's countable functionals). *For each n, we define a set C_n and a notion of* associate *for the elements of C_n as follows*:

- *Take $\mathsf{C}_0 = \mathbb{N}$, $\mathsf{C}_1 = \mathbb{N}^{\mathbb{N}}$, and declare each $f \in \mathsf{C}_1$ to be an associate for itself.*

- *For $n \geq 1$, say $f \in \mathsf{C}_1$ is an associate for the function $F\colon \mathsf{C}_n \to \mathbb{N}$ iff whenever g is an associate for $G \in \mathsf{C}_n$ we have $f \mid g = F(G)$; and take C_{n+1} to be the set of functions $\mathsf{C}_n \to \mathbb{N}$ that have an associate.*

This is essentially an extensional collapse construction (Definition 1.2; see also Section 5.1). Kleene extended the above definition from pure types to arbitrary types by means of standard type-changing techniques, giving rise to a type structure C.[11] This structure is called ECF in Troelstra [1973, §2.6].

In Kreisel's definition, continuity was built in via the concept of *neighbourhoods*. In any type structure A over $\mathbb{N}$, one may define for each type σ a system $\mathcal{N}_\sigma$ of subsets of A_σ as follows:

- $\mathcal{N}_0$ consists of all singletons $\{n\}$.
- $\mathcal{N}_{\sigma\to\tau}$ consists of all sets of the form

$$[U_1 \mapsto V_1] \cap \ldots \cap [U_r \mapsto V_r] \; (r \geq 1),$$

  where $U_i \in \mathcal{N}_\sigma$, $V_i \in \mathcal{N}_\tau$, and $[U \mapsto V]$ means the set

$$\{f \in A_{\sigma\to\tau} \mid \forall x \in U.\, f \cdot x \in V\}.$$

The elements of $\mathcal{N}_\sigma$ are called *neighbourhoods* and correspond to finite pieces of information about elements of A_σ. Clearly, any basic neighbourhood may be denoted by a formal expression involving $\mapsto, \cap$, and singletons, and obviously one can define such "formal neighbourhoods" quite independently of any type structure.

Kreisel gave a syntactical construction of C in terms of these formal neighbourhoods and functions over them: elements of $\mathsf{C}_{\sigma\to\overline{0}}$ were first given by functions on formal neighbourhoods of type σ, and only then turned into functions on the elements of C_σ themselves. It was thus automatic that all the functionals in C were continuous in the sense determined by the neighbourhood systems. However, the precise details are a little complicated and we will omit them here.

The equivalence between the Kleene and Kreisel definitions, was recognized at the time of their discovery and is mentioned in both of the original papers. (However, the proof turned out to be less trivial than these authors initially suspected; see also Hinata and Tugué [1969].) Tait (Tait [1962]) later gave a somewhat cleaner, more axiomatic treatment of the continuous functionals in the spirit of Kreisel's definition.

Both Kleene and Kreisel identified a natural effective substructure RC of C, consisting of the *recursively countable* or *recursively continuous* functionals. In Kleene's terms, these are the functionals with at least one recursive associate;

[11]Actually, Kleene first defined the countable functionals to be certain elements of S, whose action on other countable functionals was as suggested by the above definition. However, the possibility of considering the countable functionals "by themselves" was also clear to him from the outset.

in Kreisel's terms, they are defined via recursive functions on bases of neighbourhoods. An important result, the *density theorem* (see Kreisel [1959, Appendix]), ensures that each $F \in \mathsf{C}_{\sigma \to \tau}$ is completely determined by its effect on the recursive (or even the "finite") elements of C_σ; it follows that RC is extensional when regarded as a type structure in its own right.

Kreisel, in particular, was interested in using C to give constructive interpretations for systems such as intuitionistic second order arithmetic. The idea is as follows: given a formula ϕ, first apply the Gödel Dialectica translation (see Section 3.1.1) to obtain a formula $\phi' \equiv \exists \boldsymbol{y}.\forall \boldsymbol{z}.\phi^*$ in which ϕ^* is quantifier-free and the $\boldsymbol{y}, \boldsymbol{z}$ may be of any finite type. We may now consider the interpretations of ϕ' arising by allowing the $\boldsymbol{y}$ and $\boldsymbol{z}$ to range over various kinds of finite type object. Kreisel suggested that we get an interpretation that fits well with the informal notion of constructive truth if we let $\boldsymbol{y}$ range over all functionals in RC and $\boldsymbol{z}$ over functionals in C. The intuition is that the constructive content of ϕ should be embodied by effectively given operations (the $\boldsymbol{y}$) which make ϕ^* true even when presented with arbitrary continuous data such as free choice sequences (the $\boldsymbol{z}$).

Kreisel also considered rival interpretations of this kind which are of interest for independence proofs: for instance, we may let $\boldsymbol{z}$ range over arbitrary functionals in S and $\boldsymbol{y}$ over Kleene computable ones; or we may let both $\boldsymbol{y}$ and $\boldsymbol{z}$ range over the hereditarily effective operations. In addition, he showed that the above interpretation in C could be combined with Gödel's double-negation translation to give a very satisfactory "no-counterexample interpretation" for *classical* second order arithmetic. The present author considers Kreisel's paper (Kreisel [1959]) to be a classic in the field, which remains well worth reading as a discussion of the metamathematical applications of higher type functionals, and which contains in seminal form an amazing number of ideas that were later to become major themes in the subject.

One can also consider the class KC(C) of Kleene computable functionals over C, as given by Definition 3.2. It was clear to Kleene and Kreisel that every total Kleene computable functional over C is a recursively continuous functional — intuitively, anything that is computable in Kleene's "extensional" sense is computable at the more intensional level of associates. Moreover, at type 2 the recursively continuous functionals clearly coincide with the Kleene computable functionals (over C or S), since *e.g.*, any $F \in \mathsf{C}_2$ is Kleene computable relative to an associate for F.

It was raised as an open problem in Kreisel [1959, §4.4] whether all elements of RC at types 3 and above were Kleene computable. This was answered negatively by Tait (Tait [1962]), who gave as a counterexample the type 3 *modulus of uniform continuity* functional Φ (also known as the *fan functional* because it embodies the constructive content of Brouwer's Fan Theorem). The existence of Φ depends on the fact that (classically) every continuous function

$F\colon 2^{\mathbb{N}} \to 2$ is uniformly continuous. To define Φ, let *binary*$(f : \overline{1})$ be the predicate $\forall x.\, f(x) = 0 \vee f(x) = 1$, and take

$$\Phi(F : \overline{2}) = \mu n. \forall f, g.\ (\mathit{binary}(f) \wedge \mathit{binary}(g) \wedge (\forall m < n.\, f(m) = g(m))) \\ \Longrightarrow F(f) = F(g).$$

Tait showed that Φ is recursively continuous but not Kleene computable over C — nor, indeed, is it the restriction to C_2 of a function Kleene computable over S. (Published proofs of Tait's result may be found in Gandy and Hyland [1977, §4] or Normann [1999a, §5].) We therefore have two distinct notions of computability for the total continuous functionals, represented by RC and $\mathsf{KC}(\mathsf{C})$ respectively.

3.3.2. *Further characterizations.* Apart from Tait's work, there seems to have been little work on the total continuous functionals during the 1960s. The 1970s, however, saw a wealth of new developments which were mostly of two kinds: results providing further characterizations of C and RC, illuminating their basic character and confirming their natural status; and results concerning various notions of relative definability and degree structures on C.

Many of the later characterizations of C were much simpler and more immediately appealing than the original definitions. Ershov (Ershov [1974a], [1977a]) showed that the continuous functionals could be obtained from the type structure P of partial continuous functionals via an extensional collapse construction (see Theorem 4.13 below). A very similar characterization of C was obtained independently by Scott, using a type structure arising from the category of algebraic lattices (see Hyland [1975]).

Another pleasing characterization of C is that based on the idea of *sequential continuity*.[12] This definition may be presented very simply as follows. We use the notation $[x_i]$ for an infinite sequence $x_0, x_1, \ldots$.

Definition 3.14 (Sequential continuity). *For each type σ, we define a set C_σ together with a relation $[x_i] \downarrow x$ (read as "$[x_i]$ converges to x") between infinite sequences and elements of C_σ:*

- *$\mathsf{C}_0 = \mathbb{N}$, and $[x_i] \downarrow x$ if $x_i = x$ for all sufficiently large i;*
- *$\mathsf{C}_{\sigma\to\tau}$ is the set of all $f\colon \mathsf{C}_\sigma \to \mathsf{C}_\tau$ such that $[f(x_i)] \downarrow f(x)$ whenever $[x_i] \downarrow x$. And $[f_i] \downarrow f$ in $\mathsf{C}_{\sigma\to\tau}$ iff $[f_i(x_i)] \downarrow f(x)$ whenever $[x_i] \downarrow x$.*

This is tantamount to the definition of the type structure over $\mathbb{N}$ in the cartesian closed category of *L-spaces* (see Kuratowski [1952]). Scarpellini considered the type structure defined in this way as a model for bar recursion at higher types (Scarpellini [1971]), apparently without realizing that it coincided with the Kleene-Kreisel continuous functionals. The equivalence was shown by Hyland (Hyland [1975], [1979]), who collected together the

[12]The terminology, which was used in the literature of the time, refers to continuity based on sequences. It should not be confused with the more modern use of the term "sequential", which we discuss in Section 4.3.

known characterizations and clarified the relationships between them. He also discovered some new characterizations of C: for instance, as the type structure over $\mathbb{N}$ in the cartesian closed category of *filter spaces*, or in the cartesian closed category of compactly generated Hausdorff spaces. (A published proof of the latter fact appeared first in Normann [1980].) Normann later gave yet another construction via a hyperfinite type structure in the sense of non-standard analysis (Normann [1983], [1999a]).

Bergstra (Bergstra [1976], [1978]) gave a interesting characterization of C as a maximal type structure (subject to some basic closure requirements) in which all type 2 functions are continuous. Intuitively, one can construct C by taking, at each type level ≥ 3, all functions that can be added without inducing any discontinuous type 2 functions. Grilliot (Grilliot [1971]) had already shown that a type 2 functional F is continuous iff ${}^2\exists$ is *not* Kleene computable relative to F and some type 1 function; we therefore have a characterization of C as a maximal type structure closed under Kleene computation and not containing ${}^2\exists$.

By this stage it was very clear that C was the canonical choice of a full continuous type structure of total functionals over $\mathbb{N}$. Moreover, Hyland's work also showed that all the constructions of C that admitted a natural effectivization gave rise to the same effective substructure, namely RC.

3.3.3. *Degrees and relative computability.* Most of the work on degrees and relative computability for C has focussed on the notions arising from relative Kleene computability. Hinman (Hinman [1973]) gave an example of an *irreducible* element of C_2 — that is, one whose Kleene degree is not the Kleene degree of any type 1 function. Hyland in Gandy and Hyland [1977, §5] gave a simpler example of this phenomenon, making use of the Kleene tree. The same paper also contained an example, due to Gandy, of a type 3 object $\Gamma \in RC_3$ which is not Kleene computable relative to the fan functional.

In view of this last result, it was natural to ask whether any good "basis"of functionals in RC could be given, relative to which all functionals in RC were Kleene computable. (A closely related question, discussed by Feferman (Feferman [1977a]), was whether one could give a definition of RC as a substructure of C via monotone inductive schemata, in the spirit of generalized recursion theory.) It is fairly easy to specify an infinite basis for RC consisting of *partial* recursively continuous functions, one for each type level (see Bergstra [1978, §1]). Normann (Normann [1979b], [1981a]) gave a similar infinite basis consisting of total recursively continuous functionals (*i.e.*, elements of RC), but showed that none of the functionals in this were Kleene computable relative to any functionals of lower type. It follows that no *finite* basis of this kind for RC is possible. (See however Theorem 4.25 for a more satisfactory ending to this story.)

Another group of results from this period concerned properties of 1-sections for continuous type 2 objects. Early results in this vein were obtained by Grilliot (Grilliot [1971]), who pointed out that such a 1-section is never closed under the ordinary jump operator. Bergstra (Bergstra [1976]) showed that there are objects $F \in \mathsf{C}_2$ whose 1-section is not the 1-section of any $f \in \mathsf{C}_1$ (this improves on the result of Hinman mentioned above). Results relating to ordinal hierarchies for 1-sections of continuous objects are obtained in Bergstra and Wainer [1977], Normann [1978a], Normann and Wainer [1980].

A simple but beautiful result of Normann (Normann [1981b]) (who traces the idea to Kreisel [1959]) reveals a connection between the continuous functionals and some classical logical complexity classes for subsets of $\mathbb{N}^{\mathbb{N}}$. A proof of the theorem also appears in Normann [1999a].

THEOREM 3.15 (Projective hierarchy). *Let $k > 0$, $A \subseteq \mathbb{N}^{\mathbb{N}}$. Then*

(i) *A is Π^1_k iff there is a primitive recursive predicate R (e.g., definable by Kleene's $S1$–$S8$ over C) such that*

$$f \in A \iff \forall G \in \mathsf{C}_k . \exists n \in \mathbb{N}. R(f, G, n).$$

(ii) *A is Σ^1_k iff there is a primitive recursive R such that*

$$f \in A \iff \forall G \in \mathsf{C}_{k+1} . \exists n \in \mathbb{N}. R(f, G, n).$$

and moreover there is a uniform algorithm (e.g., a Kleene computable partial functional over C) which given any $f \notin A$ returns a G such that $\forall n. \neg R(f, G, n)$.

Normann showed that many facts about 1-sections and 2-envelopes flow from this theorem.

Finally, we mention that an alternative degree structure on C can be obtained by considering a more generous notion of relative computability: take $F \preceq_c G$ iff there is a recursive type 2 functional which transforms any associate for G into an associate for F. Most of the known results about degrees, sections and envelopes with respect to $\preceq_c$ are collected in Hyland [1978].

Normann's book (Normann [1980]) contains most of what was known about the total continuous functionals by 1980.

3.3.4. *Recent work.* Work on the continuous functionals abated again during the 1980s, but was renewed in the 1990s, partly owing to interest within the computer science community. Modern treatments have tended to favour versions of the Ershov-Scott construction of C (see Theorem 4.13) via domains or information systems (Berger [1993], Stoltenberg-Hansen, Lindström, and Griffor [1994], Schwichtenberg [1996], Normann [1999a]). A particular focus of recent work has been the search for abstract formulations of the concept of *totality* for elements of such domains. Berger (Berger [1993]) has demonstrated the significance of the dual notions of *density* and *codensity* (= totality) for subsets of domains, and given generalized versions of the density theorem and Kreisel-Lacombe-Shoenfield theorem in a domain-theoretic framework. A related approach to totality is pursued in Normann [1989], [1997]. More recent work has been concerned with extending the construction of C and the associated results on domains, density and totality to transfinite types and Martin-Löf style dependent types with universes (Berger [1997], Kristiansen and Normann [1997]). Bauer and Birkedal (Bauer and Birkedal [2000]) have shown how much of this material fits smoothly into the framework of Scott's *equilogical spaces* (see Section 5.2).

Some other recent results involving C (Normann [2000], Plotkin [1997]) will be mentioned in Section 4.3.4 below.

3.3.5. *Type two effectivity.* It is convenient at this point to mention briefly the work of Weihrauch and his colleagues (see *e.g.*, Weihrauch [1985], [2000b]), who have developed a framework for computable analysis known as *type two effectivity*. This framework was proposed independently of the work on C that we have described, but there are close connections. Weihrauch considers a model of computation consisting of Turing machines with infinite input and output tapes; these essentially compute uniformly partial recursive type 2 functionals on $\mathbb{N}^{\mathbb{N}}$ (Definition 2.2), subject to some minor caveats. For the most part, Weihrauch's theory is concerned with the use of this model to represent computability on other sets, such as spaces arising in analysis. A *representation* of a set S is a partial function from $\mathbb{N}^{\mathbb{N}}$ onto S — for instance, we might represent the reals by (coded) Cauchy sequences of rational approximations.

The key result which makes the theory work (presented in Weihrauch [1985]) is that the set of continuous functions on $\mathbb{N}^{\mathbb{N}}$ may itself be represented in this way by $\mathbb{N}^{\mathbb{N}}$. One obtains an effective version of the theory by restricting here to functions represented by $\mathbb{N}^{\mathbb{N}}_{rec}$. In essence, the idea here is the same as the idea behind Kleene's definition of associates (Definition 3.13), which provides representations of just this kind for the total continuous functionals. These and other connections have recently been made precise in Bauer [2001] (see also Section 5.2 below).

3.4. The hereditarily effective operations. As we have seen, one notion of total computable functional of higher type is given by the effective submodel RC of the total continuous functionals; here we have computable functionals acting on (possibly arbitrary) continuous data. One might also ask if there are interesting type structures based on the idea of computable functions acting only on computable data. Several ways of constructing such a type structure might suggest themselves. For instance:

- We might consider higher type generalizations of the definition of type 2 effective operations based on recursive indices (see Definition 2.3).
- We might consider effective *analogues* of various definitions of C: that is, we might mimic some construction of C taking only the effective functionals at each type level.

Both kinds of construction were considered in Kreisel [1959, §4.2]. As an instance of the first kind, Kreisel defined an extensional type structure HEO. Note that the definition is a straightforward extensional collapse construction:

DEFINITION 3.16 (Hereditarily effective operations). *For each type σ, define a partial equivalence relation $\equiv_\sigma$ on $\mathbb{N}$ as follows:*

- $x \equiv_0 x'$ *iff* $x = x'$;
- $x \equiv_{\sigma\to\tau} x'$ *iff for all* $y \equiv_\sigma y'$ *we have* $\phi_x(y) \equiv_\tau \phi_{x'}(y')$.

Now let HEO_σ *be the set of* $\equiv_\sigma$*-equivalence classes. Since there are well-defined total application operations* $\mathsf{HEO}_{\sigma\to\tau} \times \mathsf{HEO}_\sigma \to \mathsf{HEO}_\tau$ *induced by recursive index application, we have a total extensional type structure* HEO *over* $\mathbb{N}$.

Kreisel also introduced (in Kreisel [1958]) a closely related non-extensional type structure:

DEFINITION 3.17 (Hereditarily recursive operations). *For each type σ, define a subset* $\mathsf{HRO}_\sigma \subseteq \mathbb{N}$ *as follows:*

- $\mathsf{HRO}_0 = \mathbb{N}$;
- $x \in \mathsf{HRO}_{\sigma\to\tau}$ *iff for all* $y \in \mathsf{HRO}_\sigma$ *we have* $\phi_x(y)\downarrow$ *and* $\phi_x(y) \in \mathsf{HRO}_\tau$.

This defines a total non-extensional type structure HRO *over* $\mathbb{N}$, *with application given by recursive index application.*

The above names, and the notations HEO, HRO, were introduced later by Troelstra, who independently rediscovered these structures around 1970 (see Troelstra [1973, §2.4.17]). Note that the structure HEO is independent (up to isomorphism) of the choice of enumeration for the partial recursive functions, whereas this is not true for HRO — hence it is misleading, strictly speaking, to talk about "the" hereditarily recursive operations.

As an instance of the second kind of construction suggested above, Kreisel proposed a recursive analogue of his construction of C via formal neighbourhoods. We give here an equivalent definition based on a recursive analogue of Kleene's construction of C via associates (Definition 3.13).

DEFINITION 3.18 (Hereditarily recursively countable functionals). *For each* n, *define a set* HRC_n *and a notion of associate for elements of* HRC_n *as follows:*

- *Take* $\mathsf{HRC}_0 = \mathbb{N}$, $\mathsf{HRC}_1 = \mathbb{N}^{\mathbb{N}}_{rec}$, *and declare each* $f \in \mathsf{HRC}_1$ *to be an associate for itself.*
- *For* $n \geq 1$, *say* $f \in \mathsf{HRC}_1$ *is an associate for* $F : \mathsf{HRC}_n \to \mathbb{N}$ *iff whenever* g *is an associate for* $G \in \mathsf{HRC}_n$ *we have* $f \mid g = F(G)$; *and take* HRC_{n+1} *to be the set of functions* $\mathsf{HRC}_n \to \mathbb{N}$ *that have an associate.*

The definition may be extended from pure types to arbitrary types by standard methods, yielding a type structure HRC. This structure is called $\mathsf{ECF}(\mathcal{R})$ in Troelstra [1973, §2.6].

Kreisel noted the remarkable fact that these two approaches to constructing a class of hereditarily computable functionals coincide — that is:

THEOREM 3.19. $\mathsf{HEO} \cong \mathsf{HRC}$.

At type 2 this is essentially the Kreisel-Lacombe-Shoenfield theorem (Theorem 2.6). The generalization to higher types was noted in Kreisel [1959, §4.2] without detailed proof. Some further details appeared in Tait [1962], but a complete published proof first appeared in Troelstra [1973, §2.6].

Some further remarks may help to clarify the relationship between HRC (the recursive *analogue* of C) and RC (the recursive *substructure* of C). To build RC, one first builds the whole of C and then extracts the effective elements. To build HRC, one considers at each type level only the effective total functionals on the set of effective objects of the next type down. Thus, for instance, the Kleene tree functional Z defined in Section 2.4 lives in HRC_2, but it is not (the restriction of) an element of RC_2 since it has no computable extension to the whole of $\mathbb{N}^{\mathbb{N}}$. We therefore have a proper inclusion of RC_2 in HRC_2. Conversely, there is no element in HRC_3 corresponding to the fan functional $\Phi \in \mathsf{RC}_3$, essentially because any computable functional $\Phi' \in \mathsf{HRC}_3$ that extended Φ (with respect to the above inclusion) would be undefined on Z. There is therefore no canonical inclusion from RC_3 to HRC_3 or *vice versa*.

The idea here is that Z and Φ cannot live together in the same universe of computable total functionals. Indeed, in Part II we will formulate and prove an "anti-Church's thesis for total functionals", to the effect that there is *no* type structure of computable functionals over $\mathbb{N}$ that subsumes both HRC and RC. This suggests that, for hereditarily total functions at least, the dichotomy between "computable acting on computable" and "computable acting on continuous" is a fundamental one.

We have seen that the recursive analogues of both Kleene's and Kreisel's definition of C yield the same type structure HRC. Later results confirmed the impression that whenever any natural definition of C admits a recursive analogue, this analogue turns out to be HRC. Several such equivalences are verified in Hyland [1975]. In addition, Ershov (Ershov [1976b], [1977a]) showed that just as the hereditarily total elements of P give rise to C (see Theorem 4.13), so the hereditarily total elements of $\mathsf{P}^{\mathit{eff}}$ give rise to HEO. Ershov also made explicit a higher type generalization of the Kreisel-Lacombe-Shoenfield theorem implicit in the fact that $\mathsf{HEO} \cong \mathsf{HRC}$: namely, that all operations in HEO are continuous in the sense of the neighbourhood structure given in Section 3.3.1.

This contrasts with the surprising fact, discovered by Gandy around 1965, that not all the functionals in HEO are *sequentially* continuous in the sense of Definition 3.14. His ingenious construction of a counterexample at type 3 appears in Gandy and Hyland [1977, §8].

Later, Bezem (Bezem [1985a]) gave another characterization of HEO as the extensional collapse of HRO (see Definition 1.2):

THEOREM 3.20. $\mathsf{HEO} \cong \mathsf{EC}(\mathsf{HRO})$.

What is perhaps surprising is that this result is decidedly non-trivial. The proof essentially goes via the isomorphism with HRC as defined via Kleene associates (see Definition 3.18).

Another descendant of Kreisel's definition of HEO was Girard's category ***PER*** of partial equivalence relations on the natural numbers (Girard [1972]). We will give the definition of this category in Section 5; meanwhile, we simply remark that HEO can be naturally seen as the finite type structure over N in the cartesian closed category ***PER***.

§4. Partial computable functionals. The picture outlined in Section 3 exhibits various oddities which arise from the fact that we are restricting ourselves to hereditarily total objects. Intuitively, any computational paradigm powerful enough to generate all total computable functions will naturally generate partial ones as well, and so any restriction to total functionals will in some ways be artificial. Moreover, this artificiality is exacerbated as one passes to higher types. The most striking instance of this is the brood of difficulties associated with Kleene's S8 (see Section 3.2.1). Another tension arising from the insistence on totality can perhaps be discerned in the incompatibility of the Kleene tree functional and the fan functional (see Section 3.4). These observations might lead one to suspect that a theory of hereditarily partial objects of higher type would work more smoothly than one for total objects.

The idea that "partial is easier than total" at higher types was first aired in Kleene [1963, §9.3], and was frequently discussed in the later literature, *e.g.*, in Platek [1966], Gandy and Hyland [1977], Feferman [1977a], Ershov [1977a]. We will see in this section that the above suspicions are amply justified — that notions of partial computable functional do indeed lead to simpler theories, and enjoy more pleasant properties, than the total notions. (This accords with our experience in ordinary recursion theory: for example, the partial recursive functions are recursively enumerable but the total recursive functions are not.) This is not to say the total notions are necessarily of lesser interest than the partial ones — we have already seen that some total type structures have very good mathematical credentials — but as we shall see, a study of the partial notions greatly enriches our understanding of the total ones.

Nevertheless, there are some important conceptual questions to be addressed in formulating notions of hereditarily partial object of higher type. For instance, should our type 1 objects be partial functions $\mathbb{N} \rightharpoonup \mathbb{N}$ (*i.e.*, functions $\mathbb{N} \to \mathbb{N}_\perp$)? Or should our treatment be so thoroughly partial that we even interpret $\overline{0}$ as $\mathbb{N}_\perp$, in which case our type 1 objects might be (for instance) functions $\mathbb{N}_\perp \to \mathbb{N}_\perp$? Questions of this kind will proliferate as we pass to higher types, and at first sight it is unclear how we should respond to them. One can speculate that a lurking unease about these issues may partly account for the relatively late development of good notions of partial higher type object.

With the benefit of hindsight, we can see that these choices are not as significant as they might seem, since all reasonable choices turn out to be "equivalent" in some sense. We will discuss these issues in detail in Part II, where a precise statement to this effect will be given. For the purposes of our historical survey, however, it will be useful to introduce here two rival definitions of "partial type structure" that both figured in the development of the subject:

DEFINITION 4.1 (Call-by-name, call-by-value).

(i) *A* call-by-name (or CBN) structure *is an extensional total type structure over* $\mathbb{N}_\perp$.
(ii) *A* call-by-value (or CBV) structure *is an extensional partial type structure over* $\mathbb{N}$ (*in the sense of Section* 1.5.1).

This terminology is borrowed from computer science, and does not appear in the earlier recursion theory literature. The significance of the terms can be appreciated by considering the nature of type 1 objects under the two definitions. In a CBN structure, a type 1 object is a function $\mathbb{N}_\perp \to \mathbb{N}_\perp$, and the argument fed to such a function is not itself a natural number but a *name* for one — that is, a computational process which may or may not evaluate to yield a natural number. In a CBV structure, a type 1 object is a functions $\mathbb{N} \to \mathbb{N}_\perp$, which demands a genuine natural number as its argument — that is, the computation of the *value* of the argument must take place before the function is called.

As we shall see in Part II, in all cases of interest CBN and CBV structures are equivalent, in the sense that from any CBN structure we may recover a corresponding CBV structure and *vice versa*. We can therefore regard corresponding CBN and CBV structures simply as different concrete presentations of the same underlying notion of computability. However, this perspective was probably not available to the pioneers of the subject.

Throughout this section, if X is a poset, we write $X_\perp$ for the poset obtained by adding a new least element $\perp$.

4.1. Partial monotone functionals. As far as we are aware, the earliest formulation of a class of hereditarily partial computable functionals at all finite types is due to Davis (Davis [1959]). Davis (together with Putnam) realized that some restriction on set-theoretic partial functionals was needed to obtain a good theory, and proposed (essentially) a call-by-name structure A of *consistent* partial functionals ($f : A_n \rightharpoonup \mathbb{N}$ is consistent if whenever $v, w \in \mathrm{dom}(f)$, $t \in A_n$ and $v \subseteq t$, $w \subseteq t$, we have $f(v) = f(w)$), and a notion of computable element of A. Certain features of Davis's definition closely foreshadow Scott's later work on domain theory (see Section 4.2); however, the consistency condition appears to be too weak to be really fruitful, and Davis's type structure seems lacking in good properties at higher types.

4.1.1. *Platek's thesis.* A very satisfactory notion of partial functional was introduced and studied by Platek in his monumental Ph.D. thesis (Platek [1966]). His first fundamental insight was that a good theory could be obtained by restricting attention to *monotone* partial functionals. Essentially, he considered recursion theory over the following call-by-value structure:

Definition 4.2 (Hereditarily monotone functionals). *For each type σ define a poset M_σ as follows: take $\mathsf{M}_0 = \mathbb{N}$ with the discrete order, and for $\sigma = \sigma_1 \to \cdots \to \sigma_r \to \overline{0}$ $(r \geq 1)$, let M_σ be the set of all monotone partial functions $\mathsf{M}_{\sigma_1} \times \cdots \times \mathsf{M}_{\sigma_r} \rightharpoonup \mathbb{N}$ endowed with the pointwise order. Application in M is defined by*

$$f \cdot x \simeq \Lambda z_2 \ldots z_r . f(x, z_2, \ldots, z_r).^{13}$$

(where $f \in \mathsf{M}_\sigma$ and $x \in \mathsf{M}_{\sigma_1}$).

The type structure S can be embedded in M via injections $\Psi_\sigma : \mathsf{S}_\sigma \to \mathsf{M}_\sigma$: take $\Psi_0(x) = x$, and for $\sigma = \sigma_1 \to \cdots \to \sigma_r \to \overline{0}$ we let $\Psi_\sigma(F)(\boldsymbol{g}) = y \in \mathbb{N}$ iff there exist $G_i \in \mathsf{S}_{\sigma_i}$ with $\Psi_{\sigma_i}(G_i) \sqsubseteq g_i$ and $F(\boldsymbol{G}) = y$. We say $f \in \mathsf{M}_\sigma$ *represents* $F : \mathsf{S}_{\sigma_1} \times \cdots \times \mathsf{S}_{\sigma_r} \rightharpoonup \mathbb{N}$ if $f(\Psi(\boldsymbol{G})) \simeq F(\boldsymbol{G})$ for all $\boldsymbol{G}$. If $F \in \mathsf{S}_\sigma$, clearly $\Psi(F)$ represents (the uncurried form of) F.

One can give a computational motivation for the restriction to monotone functionals: if f is any computable partial functional, we would expect an increase in the available information about the input to f to allow (if anything) an increase in the output information produced by f. Platek's main motivation, however, was to provide a setting in which definitions by recursion make sense. Given any recursion equation

$$f(x) = F(f, x)$$

where $x : \sigma, f : \sigma \to \tau$ are variables and $F \in \mathsf{M}_{(\sigma\to\tau)\to\sigma\to\tau}$, Tarski's fixed point theorem ensures that there is a unique least element $f \in \mathsf{M}_{\sigma\to\tau}$ satisfying the equation for all $x \in \mathsf{M}_\sigma$. This element f may be obtained via a transfinite iteration of F:

$$f_0(x) = \bot, \quad f_{\alpha^+}(x) = F(f_\alpha, x), \quad f_\lambda = \bigcup_{\alpha<\lambda} f_\alpha, \quad f = \bigcup f_\alpha$$

(where λ ranges over limit ordinals). As mentioned earlier, Platek was more concerned with questions of *definability* than of effective computability, so the transfinite nature of the computation here was not seen as a problem.

[13]Platek used the term *consistent* rather than *monotone*. The structure M can be construed as a call-by-value structure in our sense, though it does not quite coincide with the "natural" full monotone call-by-value structure over $\mathbb{N}$, for which one would take $\mathsf{M}_{\sigma\to\tau}$ to be the set of monotone partial functions $\mathsf{M}_\sigma \rightharpoonup \mathsf{M}_\tau$. However, for pure types the two definitions coincide exactly, and the minor difference can be glossed over.

Platek actually defined such a structure relative to an arbitrary set of objects in place of $\mathbb{N}$, since one of his aims was to give a uniform treatment of recursion theory which applied also to other domains, such as ordinals and transitive sets.

Platek's second fundamental insight was that definitions by recursion can be conveniently expressed by means of *fixed point operators*. Specifically, for each type $\rho = \sigma \to \tau$ there is an element $Y_\rho \in \mathsf{M}_{(\rho\to\rho)\to\rho}$ such that for all $F \in \mathsf{M}_{\rho\to\rho}$, $Y_\rho(F)$ is the unique least element $f \in \mathsf{M}_\rho$ such that $f(x) = F(f, x)$ for all x.

Other (much simpler) ways of explicitly defining new elements of M from old ones are encapsulated by the elements $I_\sigma, K_{\sigma\tau}, S_{\rho\sigma\tau}, D \in \mathsf{M}$ defined by

$$\begin{aligned}
I_\sigma(x^\sigma) &= x, \\
K_{\sigma\tau}(x^\sigma, y^\tau) &= x, \\
S_{\rho\sigma\tau}(f^{\rho\to\sigma\to\tau}, g^{\rho\to\sigma}, x^\rho) &\simeq (fx)(gx), \\
D(x^{\overline{0}}, y^{\overline{0}}, f^{\overline{1}}, g^{\overline{1}}) &= \begin{cases} f & \text{if } x = y, \\ g & \text{otherwise.} \end{cases}
\end{aligned}$$

(Here D stands for "definition by cases".) These lead us to the following definition of recursive definability over M:

Definition 4.3. *Given any set $B \subseteq \mathsf{M}$, define the set $\mathcal{R}(B)$ of elements* recursively definable from *B to be the smallest subset of* M *containing B and all the elements Y, I, K, S, D, and closed under application.*

In the case of the monotone type structure over $\mathbb{N}$, we will usually be interested in the case where B contains "enough" basic computable elements (zero, successor and predecessor suffice[14]). We say an element of M is *recursively definable* if it is recursively definable from these basic elements alone.

Platek showed that the above definition can also be cast in terms of schemata in the spirit of Kleene's S1–S9, but with an important difference: *any* definition of a computation written using these schemata has a meaning in M. Thus the theory does not suffer from the difficulties arising from the infinitary side-condition in S8.

Moreover, one can recover Kleene's notion of computability over S as follows: any partial function $F : \mathsf{S}_{\sigma_1} \times \cdots \times \mathsf{S}_{\sigma_r} \rightharpoonup \mathbb{N}$ is Kleene computable iff it is represented by some recursively definable $f \in \mathsf{M}$. Using this as an alternative definition of Kleene computability, Platek was able to obtain clearer proofs of many of Kleene's results plus some new ones, such as the fact that the first recursion theorem holds subject to a type level restriction (see Platek [1966, p. 123]). However, the equivalence with Kleene's definition is non-trivial and depends on the difficult substitution theorems of Kleene [1959b]; this seems to reflect the unmanageability of the schemata S1–S9 (see discussions in Platek [1966, pp. 114–5, 168–9]).

Platek was also the first to draw attention to the famous *parallel or* function (called *strong or* by Platek) and the issues it raises (pp. 127–131). Let us

[14]Gandy pointed out that in fact zero and successor suffice in Platek's setting.

suppose $0 \in \mathbb{N}$ stands for "true" and any other natural number stands for "false". The parallel or function may then be defined as follows:

DEFINITION 4.4. *Let por* $\in \mathsf{M}_{\overline{0}\to\overline{0}\to\overline{0}}$ *be the element given by*

$$por(x, y) = \begin{cases} 0 & \textit{if } x \textit{ is true}, \\ 0 & \textit{if } y \textit{ is true}, \\ 1 & \textit{if } x \textit{ and } y \textit{ are both false}, \\ \bot & \textit{otherwise}. \end{cases}$$

Platek pointed out that *por* is not recursively definable, although it is definable in an Herbrand-Gödel style equation calculus, and is arguably computable if we allow a kind of non-determinism in computations with regard to the order of evaluation of arguments. As we shall see below, the dichotomy between these notions of computability was to become a major theme in later work.

Platek also gave a characterization of recursive definability in terms of a formal language based on the λ-calculus, closely foreshadowing PCF (see Section 4.2.4). Rather curiously, the language he introduced was an *untyped* λ-calculus — this allowed him to make use of the counterparts of Y and D as well as I, K, S in pure λ-calculus. Platek's calculus also featured some elaborate machinery for allowing a mixture of syntax and semantics in computations — *e.g.*, a constant $\lceil f \rceil$ for every element $f \in \mathsf{M}$ was admitted, and the definition of the evaluation relation $U \Downarrow n$ included rules of the following kind (related to Kleene's S8):

$$\begin{array}{rl} \text{if} & f \in \mathsf{M}_{(\sigma\to\overline{0})\to\overline{0}},\ g \in \mathsf{M}_{\sigma\to\overline{0}} \\ \text{and} & \text{for all } h \in \mathsf{M}_\sigma \text{ such that } g(h) \in \mathbb{N},\ U\lceil h \rceil \Downarrow g(h) \\ \text{then} & \lceil f \rceil U \Downarrow f(g). \end{array}$$

The infinitary character of such rules means that computations can be of transfinite length, although for terms containing no constants of type level 2 or above, computations are always finite.

One of the major results of Platek [1966] is the following, proved under mild conditions on B:

THEOREM 4.5. *An element* $f \in \mathsf{M}_{\sigma_1\to\cdots\to\sigma_r\to\overline{0}}$ *is recursively definable from* $B \subseteq \mathsf{M}$ *iff* f *is* λ-definable *from* B, *in the sense that there is a* λ*-term* U, *containing no constants except those corresponding to elements of* B, *such that for all* $g_i \in \mathsf{M}_{\sigma_i}$ *we have* $U\lceil g_1 \rceil \ldots \lceil g_r \rceil \Downarrow n$ *iff* $f(\mathbf{g}) = n$.

The forward implication here is easily shown by supplying lambda terms corresponding to I, K, S, Y, D. To show the converse, Platek proved the existence of recursively definable *enumerators* $E_\sigma \in \mathsf{M}_{\overline{0}\to\sigma}$ such that if U λ-defines $f \in \mathsf{M}_\sigma$ then $E_\sigma(\#U) = f$ (where $\#U$ is a Gödel-number for U). Essentially the same result and proof were rediscovered in Longley and Plotkin [1997].

In summary, Platek's thesis was a major achievement which both shed considerable light on Kleene's earlier work on S1–S9 computability, and introduced many of the fundamental ideas explored in later work on PCF. Indeed, his recursively definable elements of M are exactly the PCF-definable ones in the sense of Scott and Plotkin; and his work shows the existence of a programming language with a *finitary* notion of computation for expressing these elements. With a little charity, one can read into the results of his Chapter 4 a proof of adequacy for M as a model of combinatory PCF (cf. Definition 4.16 and Theorem 4.19 below).

Platek's thesis is unfortunately not generally available, but a detailed published account of much of the material appears (in a somewhat streamlined form) in Moldestad [1977].

4.1.2. *Kleene's later work.* In his later years, Kleene returned to the study of higher type computability and published a series of papers under the title "Recursive functionals and quantifiers of finite types revisited" (Kleene [1978], [1980], [1982], [1985], [1991]). Of these papers, Kleene [1985] provides the best introduction to the whole series.

Kleene's motivation was spelt out in Kleene [1978, §1.2]:

> I aim to generate a class of functions ... which shall coincide with all the partial functions which are "computable" or "effectively decidable", so that Church's 1936 thesis (IM §62) will apply with the higher types included (as well as to partial functions, IM p. 332).

(This has been called "Kleene's problem" in Hyland and Ong [2000].) Like Platek, Kleene sought to avoid the anomalies of the original S1–S9 theory by developing a theory of hereditarily partial objects. However, whereas Platek was interested primarily in questions of definability, Kleene was more interested in the concrete nature of computations at higher types.

Kleene concentrated mostly on the type levels $j = 0, 1, 2, 3$. In Kleene [1978] he introduced a collection of schemata for defining a class of computable functions: a group of schemata S0–S7 (similar in spirit to the earlier S1–S8 though different in a few details), together with a new schema S11 to take over the role of S9. (The label S10 was used in Kleene [1959b] for the schema of μ-recursion.) We adapt Kleene's formulation slightly, for consistency with the notation of Definition 3.2:

S11: *General recursion:*

$$\{\langle 11, \#\boldsymbol{\sigma}, g\rangle\}(\boldsymbol{x} : \boldsymbol{\sigma}) \simeq \{g\}(\Lambda \boldsymbol{x}.\, \{\langle 11, \#\boldsymbol{\sigma}, g\rangle\}(\boldsymbol{x}), \boldsymbol{x}).$$

(More briefly, $\{h\}(\boldsymbol{x}) \simeq \{g\}(\{h\}, \boldsymbol{x})$ where $h = \langle 11, \#\boldsymbol{\sigma}, g\rangle$).

Whereas in the earlier theory the schema S9 had incorporated the second recursion theorem as a defining principle, giving rise to restricted versions of the first recursion theorem as consequences, the new schema S11 has the effect of building in a natural formulation of the first recursion theorem, and closure under S9 can be derived as a consequence.

In order to give an interpretation of these schemata which makes sense of the apparent circularity in S11, Kleene gave in Kleene [1978] a syntactical description of computations as computation trees labelled by formal expressions. (Kleene uses the term *j-expression* for a formal expression of type $\overline{j}$.) The essential difference between these and the computation trees of Kleene [1959b], [1963] is that in order to evaluate an expression such as $f(\lambda x.g^1(x), y^0)$ (where f, g are themselves defined via the schemata), we are no longer required to launch a subcomputation to compute the whole graph of $\Lambda x.g^1(x)$, but instead may plough on with the main computation, carrying around the $\lambda x.g(x)$ as an unevaluated formal expression. Only if we later require a particular value for g, such as $g(5)$, do we engage in a computation for g. Computation therefore takes on a much more syntactic character: we are really computing with formal expressions rather than with the semantic objects they denote, and so it need not matter that not every formal expression denotes a semantic object.

In general, the formal expressions in Kleene's computations are allowed to contain free variables of type levels $0, 1, 2, 3$: computations then proceed relative to an interpretation of these free variables as semantic objects. For this reason, a successful computation tree for a 0-expression (that is, one that results in a numerical value) may be infinitely branching and hence of transfinite depth: for example, if we wish to compute $F^2(b)$ where F^2 is a free variable and b is a 1-expression, we really do need to compute $b(n)$ for every numeral n. However, a successful computation tree for a *closed* 0-expression will always be a finite object. Indeed, Kleene's definition of computations closely foreshadows the operational definition of PCF — though as in his earlier work on S1–S9, his seeming reluctance to make fuller use of the typed λ-calculus is puzzling.

In Kleene [1980] and the following papers, Kleene proceeded to develop a semantics for his formal expressions which allows us to give an interpretation for the objects considered at all stages during a computation. In effect, Kleene constructed (the first few levels of) a call-by-name type structure U consisting of what he termed the *unimonotone* functionals (*monotone* with a *un*ique and *i*ntrinsically determined basis). Here one takes U_0 to be $\mathbb{N}_\perp$ and U_{j+1} to be a certain set of monotone functions from U_j to U_0, ordered pointwise. As in Platek's work, the monotonicity reflects the computational intuition that if we are able to obtain some output value without seeing a certain piece of input information, this value should not be affected if we later see it. Furthermore, Kleene restricted attention to those monotone functionals that could be computed by an oracle who followed a strategy or protocol of a certain kind. Kleene gave detailed explicit descriptions of the possible behaviours of such oracles at types 2 and 3, which foreshadow later work on game semantics for PCF. However, Kleene's definitions, couched in terms of the colourful imagery of oracles who open envelopes containing other oracles, were somewhat lacking in clarity and mathematical crispness, and moreover the generalization to higher types was left unclear. It seems that the expression of Kleene's ideas might have been facilitated by some of the basic concepts of domain theory and even category theory which were then available.

Nevertheless, several interesting ideas emerged from Kleene's work. For instance, Kleene isolated an important feature of "sequential" strategies for oracles: a type 2 oracle O, unless she computes a constant type 2 function, must be able to produce of her own accord a type 0 object x (*i.e.*, an element of $\mathbb{N}_\perp$) to serve as the first object she feeds to her type 1 argument f (so that if $f(x) = \perp$ then the whole computation hangs up). Kleene originally conjectured that a similar property would hold at type 3: that a non-constant type 3 oracle would always be able to come up with a particular type 1 object to serve as the first object fed to its type 2 argument. However, this was refuted by Kleene's student Kierstead, who gave as a simple counterexample the type 3 functional defined by

$$\Psi(F^2) = F(\Lambda x^0.\, F(\Lambda y^0.x)).$$

Intuitively, the first object fed to the argument F here is the type 1 function $\Lambda x.F(\Lambda y.x)$, but this function is itself dependent on information about F which only emerges later in the dialogue. This example shows that the possible interactions between oracles and their arguments are somewhat more subtle than Kleene had at first envisaged.

In the subsequent papers, Kleene was able to draw his description of type 3 oracles to a satisfactory completion. An important feature of Kleene's semantics is its intensional character: his oracles work by acting on other oracles (who here are "computing agents" with a certain behaviour) rather than on pure function extensions. Indeed, the requirement that oracles behave extensionally (that is, give the same result when presented with two different oracles for the same function) has to be explicitly incorporated into the definition at certain points. This intensional character is also an important aspect of much of the work in computer science on the semantics of higher type computation: for instance, Berry and Curien's sequential algorithms model (Berry and Curien [1982]), or the game models of PCF (see Section 4.3.3).

Kleene's unimonotone semantics can be considered as one in which computable objects act only on computable objects, if "computable" here is understood to mean realized by some oracle. However, the behaviour of oracles is not required to be effectively given, and so unimonotone functionals need not be computable in a genuinely effective sense. Even more strikingly, they need not be continuous: in fact, Kleene seems to have made a quite deliberate decision to allow infinite computation trees, and moreover to allow the behaviour of oracles to depend on infinitely much information about their arguments. In this regard, Kleene's work contrasts sharply with most work in the computer science tradition; indeed, from a modern perspective, Kleene's decision to impose such tight and computationally motivated constraints in the direction of unimonotonicity but not in the direction of continuity stands as something of a curiosity.

In the meantime, Kierstead (Kierstead [1980], [1983]) developed an alternative semantics for Kleene's S0–S7 and S11, closer in spirit to Platek's work in that it makes use of the full monotone type structure over $\mathbb{N}_\perp$. Many of Kierstead's results are closely parallel to Platek's: for instance, he embeds the total set-theoretic type structure S in the monotone one in a way which respects Kleene computability, and from this is able to deduce certain substitution properties for the total Kleene computable functionals. One difference is that Kierstead considers the call-by-name monotone type structure whereas Platek considers the call-by-value one — this appears to be correlated to some difference in the conceptual status of the models in the two approaches, but from a purely technical point of view it does not seem to be a major difference. (See also Draanen [1995] for some further information in this area.)

The present author feels that Kleene's work in this area contributed some important ideas, although many of these have been more clearly expressed and more fully explored in the subsequent computer science literature. Some useful work by Bucciarelli (Bucciarelli [1993a], [1993b]) makes explicit the relationships between Kleene's work and subsequent developments in computer science, and in particular explores the connections between Kleene's oracles and Berry-Curien sequential algorithms (see Section 4.3). The influence of Kleene's ideas on the computer science tradition is also discussed in Hyland and Ong [2000].

4.2. Partial continuous functionals. Undoubtedly one of the most compelling notions of higher type computability was discovered independently by Scott (Scott [1969], [1993]) and Ershov (Ershov [1972], [1973b]). Coming from somewhat different directions, both these authors arrived at the same type structure P of *partial continuous functionals* and an effective substructure $\mathsf{P}^{\mathit{eff}}$ of partial computable functionals.

For expository convenience we will break slightly with chronological order, describing first the work of Scott, Ershov and others on purely mathematical characterizations of these type structures, and then returning in Section 4.2.4 to consider the connections with languages such as PCF, starting again with the work of Scott in 1969. This will allow us to avoid breaking up our account of the developments relating to PCF, which will be central to much of the rest of the paper.

4.2.1. *Scott's approach.* Scott's work brings together Platek's idea of using partial functionals in recursion theory, and the Kleene-Kreisel idea of using continuity to capture the finitary aspect of computations. This leads to a theory of computable functionals which is arguably simpler and more natural than either Platek's or Kleene and Kreisel's. Scott was also motivated by the idea of giving a mathematical theory of the meanings of computer programs — an abstract view of the mathematical functions they compute as distinct from a purely machine-oriented account of how they behave — and his work lies at the root of the modern computer science tradition in denotational semantics.

The simplest definition of the partial continuous functionals was formulated in Scott [1993][15] in terms of *complete partial orders*:

Definition 4.6 (CPOs).

(i) *A* complete partial order (CPO) *is a poset* $(X, \sqsubseteq)$ *with a least element, in which every chain* $x_0 \sqsubseteq x_1 \sqsubseteq \dots$ *has a least upper bound* $\bigsqcup x_i$.
(ii) *A function* $f : X \to Y$ *between CPOs is* continuous *if* f *is monotone and for every chain* $x_0 \sqsubseteq x_1 \sqsubseteq \dots$ *in* X *we have* $f(\bigsqcup x_i) = \bigsqcup f(x_i)$.

Definition 4.7 (Partial continuous functionals).
Define a type structure P *as follows: let* P_0 *be* $\mathbb{N}_\perp$ *(considered as a CPO with the usual partial ordering), and* $\mathsf{P}_{\sigma\to\tau}$ *is the CPO of continuous functions* $f : \mathsf{P}_\sigma \to \mathsf{P}_\tau$, *ordered pointwise.*

Thus, P is simply the natural type structure over $\mathbb{N}_\perp$ in the cartesian closed category of CPOs.

Scott showed in Scott [1969] that an intrinsic notion of computability in P can be given, by exploiting the fact that the CPOs P_σ are all *domains*. Intuitively, a domain is a CPO in which there is a good notion of "finite piece of information" about an element, and in which every element is determined by the set of finite pieces of information about it. The following definitions were introduced in Scott [1969] — for more details see any standard text on domain theory, *e.g.*, Stoltenberg-Hansen, Lindström, and Griffor [1994]. (A detailed understanding of these definitions will not be required for what follows.)

Definition 4.8 (Scott domains).

(i) *A subset* D *of a poset* $(X, \sqsubseteq)$ *is* directed *if it is non-empty and for all* $x, y \in D$ *there exists* $z \in D$ *with* $x \sqsubseteq z$ *and* $y \sqsubseteq z$. *A* DCPO *is a poset* $(X, \sqsubseteq)$ *with a least element* $\perp$ *in which every directed subset* D *has a least upper bound* $\bigsqcup D$. *A function* $f : X \to Y$ *between DCPOs is a* continuous map *if it is monotone and preserves lubs of directed sets.*
(ii) *In a DCPO* X, *we say* x, y *are* consistent *(and write* $x \uparrow y$*) if they have an upper bound in* X. *A DCPO* X *is* consistently complete *if whenever* $x, y \in X$ *have an upper bound, they have a least upper bound* $x \sqcup y$.

[15]This paper was written in 1969 and circulated widely in manuscript form. It was eventually published in the Böhm Festschrift in 1993, along with illuminating retrospective comments by Scott himself.

(iii) *An element e of a DCPO X is* finite *if whenever D is directed and $e \sqsubseteq \bigsqcup D$, we have $e \sqsubseteq d$ for some $d \in D$. We write F_x for the set of finite elements $e \sqsubseteq x$.*
(iv) *A DCPO is* algebraic *if for every element x, F_x is directed and has lub x. A DCPO is* ω-algebraic *if it is algebraic and its set of finite elements is countable.*
(v) *A* domain *is a consistently complete, ω-algebraic DCPO.*

Scott showed that the CPOs P_σ are all domains, and moreover that in each P_σ one can give an *effective enumeration* $e_0, e_1, \ldots$ of the finite elements, in such a way that the relations $e_i \uparrow e_j$ and $e_i \sqcup e_j = e_k$ are semirecursive in i, j, k. We may then call an element $x \in \mathsf{P}_\sigma$ *effective* if the finite pieces of information about x are recursively enumerable — that is, if $\{i \mid e_i \sqsubseteq x\}$ is r.e. The effective elements of P are closed under application and so constitute a substructure $\mathsf{P}^{\mathit{eff}}$.

In subsequent papers (Scott [1970], [1972], [1976]), Scott shifted his attention from domains to *complete lattices*, which give rise a very similar theory but using a rather simpler and more familiar definition. Although this approach is very elegant, it can be argued that it takes us further away from computationally meaningful structures: for instance, even the type of natural numbers now needs to be represented by a poset with a top element, which does not typically correspond to the behaviour of any program (see Section 4.2.4). In the 1980s Scott returned to the original domain-theoretic ideas and gave a more concrete presentation of them in terms of *information systems* (Scott [1982]), in which the finite elements (finite pieces of information) were taken as primary.

4.2.2. *Ershov's approach.* In the meantime, Ershov had given a construction of $\mathsf{P}^{\mathit{eff}}$ of a quite different character, arising from his theory of *enumerated sets*. This theory offers a framework for studying a wide range of mathematical structures from an algorithmic or recursion-theoretic point of view (for a recent survey and further references, see Ershov [1999]). The basic definitions are as follows:

Definition 4.9 (Enumerated sets).

(i) *An* enumerated set *is simply a set X equipped with a total function $\nu\colon \mathbb{N} \to X$ (called an* enumeration*). If $\nu(n) = x$, we may say that n is a* code *or* recursive index *for x.*
(ii) *A* morphism *from (X, ν) to (X', ν') is a function $\phi\colon X \to X'$ such that there exists a recursive function f satisfying $\phi \circ \nu = \nu' \circ f$.*

In Ershov [1971b], [1971a] Ershov introduced the category ***EN*** of enumerated sets, and considered the question of when the set of morphisms from (X, ν) to (X', ν') can be endowed with an enumeration making it into the category-theoretic exponential $(X', \nu')^{(X,\nu)}$. In particular, he defined certain classes $\mathcal{C}_2, \mathcal{C}_{20}$ of enumerated sets (with $\mathcal{C}_{20} \subseteq \mathcal{C}_2$), and proved:

THEOREM 4.10. *If $E \in \mathcal{C}_2$ and $E' \in \mathcal{C}_{20}$, then the exponential ${E'}^E$ exists in **EN** and belongs to $\mathcal{C}_{20}$.*

We omit here the somewhat technical definitions of $\mathcal{C}_2$ and $\mathcal{C}_{20}$, which are reproduced *e.g.*, in Ershov [1999]. Since $\mathbb{N}$ (with the identity enumeration) is in $\mathcal{C}_2$ and $\mathbb{N}_\perp$ (with an obvious enumeration) is in $\mathcal{C}_{20}$, it follows that we can construct both call-by-name and call-by-value type structures by repeated exponentiation in ***EN***. Significantly, these type structures are constructed purely out of computable objects acting on computable objects — no notion of continuity is involved in the definition.

In Ershov [1972] Ershov gave a topological description of these structures using the concept of *f_0-spaces*:

DEFINITION 4.11 (f_0-spaces).

(i) *For any T_0 topological space X, let $\sqsubseteq_X$ be the partial order defined by*

$$x \sqsubseteq_X y \iff \textit{for every open set } V, \textit{ if } x \in V \textit{ then } y \in V.$$

Call an open non-empty set V an f-set *if it contains a least element with respect to $\sqsubseteq_X$.*

(ii) *An* f_0-space *is a T_0 space X in which the family of f-sets, together with the empty set, is closed under binary intersections and forms a basis for the topology on X, and moreover the whole of X is an f-set.*

(iii) *Let us write X_0 for the set of elements $x \in X$ for which $\{y \mid x \sqsubseteq_X y\}$ is open. We say $I \subseteq X_0$ is an* ideal *if it is downward closed under $\sqsubseteq_X$ and every pair of elements in I has a least upper bound in I. An f_0-space X is* complete *if $(X, \sqsubseteq_X)$ is canonically isomorphic to the set of ideals in X_0 ordered by inclusion.*

Ershov showed, in effect, that the category of complete f_0-spaces and continuous maps is cartesian closed, and moreover that a substructure of "effective elements" in the type structure over the complete f_0-space $\mathbb{N}_\perp$ coincides with the type structure over $\mathbb{N}_\perp$ in ***EN***. It is natural to think of this as a higher type generalization of the Myhill-Shepherdson theorem (Theorem 2.5).

In Ershov [1972, Section 8] Ershov noted in passing that this class of computable functionals had many pleasing recursion-theoretic properties, remarking that

> It is fully justified to consider [this class] as the most natural generalization (more precisely, extension) of the class of partially recursive functions.

Thus far Ershov's work had proceeded independently of Scott's, but the connections were made explicit in Ershov [1973b]. Indeed, it is not hard to show that domains and complete f_0-spaces are essentially equivalent, and that Ershov's type structures coincide exactly with P and $\mathsf{P}^{\mathit{eff}}$. Ershov's generalization of the Myhill-Shepherdson theorem may therefore be recast as follows:

THEOREM 4.12. $\mathsf{T}(\boldsymbol{EN}, \mathbb{N}_\perp) \cong \mathsf{P}^{\mathit{eff}}$.

In general, Ershov's definitions made greater use of topological concepts whereas Scott's had emphasized the order-theoretic ideas, although in fact both authors made good use of the interplay between the two perspectives.

In Ershov [1974a], [1977a] Ershov obtained the following relationship between P and the Kleene-Kreisel type structure C:

THEOREM 4.13 (Hereditarily total elements). *Define a substructure $A \subseteq \mathsf{P}$ as follows: $A_0 = \mathbb{N}$, and $A_{\sigma\to\tau} = \{f \in \mathsf{P}_{\sigma\to\tau} \mid \forall x \in A_\sigma . f \cdot x \in A_\tau\}$. Define total equivalence relations $\sim_\sigma$ on A_σ as follows: $x \sim_0 y$ iff $x = y$; and $f \sim_{\sigma\to\tau} g$ iff $\forall x \in A_\sigma . f \cdot x \sim_\tau g \cdot x$. Then the sets $A_\sigma / \sim_\sigma$ constitute a type structure that is canonically isomorphic to* C.

Moreover, if $A_\sigma^{\mathit{eff}} = A_\sigma \cap \mathsf{P}_\sigma^{\mathit{eff}}$, the sets $A_\sigma^{\mathit{eff}} / \sim_\sigma$ constitute a type structure isomorphic to RC.

It follows easily that C is the extensional collapse of P with respect to the equality relation on $\mathbb{N}$. However, the above theorem says more than this, since it tells us that all hereditarily total elements are automatically hereditarily extensional. An analogous result for effective type structures, implying that the extensional collapse of $\mathsf{P}^{\mathit{eff}}$ gives rise to HEO, was proved in Ershov [1976b].

Ershov's main results on P and $\mathsf{P}^{\mathit{eff}}$ are conveniently summarized in Ershov [1977a]. Some further discussion of the relationship between the partial and total type structures also appears in Sections 9 and 10 of Gandy and Hyland [1977]. A streamlined presentation of the main results on f_0-spaces appears in Giannini and Longo [1984], together with some applications to the semantics of *untyped* λ-calculi.

4.2.3. *Later developments.* An important structural property of both P and $\mathsf{P}^{\mathit{eff}}$ was obtained in Plotkin [1978]. This makes use of the notion of a *coherent* Scott domain, that is, one in which every pairwise consistent subset has a least upper bound.

THEOREM 4.14. *The Scott domain $\mathbb{T}^\omega$ of functions from $\mathbb{N}$ to $2_\perp$ ordered pointwise is a* universal *coherent domain: that is, for any coherent Scott domain X there are continuous maps $f_X : X \to \mathbb{T}^\omega$ and $g_X : \mathbb{T}^\omega \to X$ such that $g_X \circ f_X = \mathrm{id}_X$ (we say X is a* retract *of $\mathbb{T}^\omega$). Moreover, if X is an effective domain then these maps are computable.*

Since all the domains P_σ are coherent, it follows that any domain that is rich enough to contain $\mathbb{T}^\omega$ as a computable retract (such as P_1) contains all the P_σ as computable retracts. Indeed, using some standard categorical techniques, the whole of P can be reconstructed from just P_1 together with its set of continuous endofunctions; likewise, $\mathsf{P}^{\mathit{eff}}$ can be reconstructed from the monoid of computable endofunctions on $\mathsf{P}_1^{\mathit{eff}}$. (For an exposition of these general techniques, see *e.g.*, Longley [2002b].)

A few later results should also be mentioned. For instance, several people noted that Ershov's category ***EN*** has rather poor closure properties (in particular, it is not cartesian closed), and so proposed larger categories to remedy this. (Of course, the category ***EN*** is still of interest, since we know more about an object by knowing it belongs to ***EN*** than by knowing it lives in some larger category.) Mulry (Mulry [1982]) showed that ***EN*** could be embedded into his *recursive topos* preserving existing exponentials, so that the type structure over a suitable object $N_\perp$ in this topos coincides with $\mathsf{P}^{\mathit{eff}}$.[16] This is close in spirit to another characterization of $\mathsf{P}^{\mathit{eff}}$ given by Longo and Moggi (Longo and Moggi [1984b]), which gives the remarkable appearance of magically dispensing with any explicit computability requirement at higher types. Notice the analogy with Definition 2.1.

Definition 4.15. *Let fst, snd*: $\mathbb{N} \to \mathbb{N}$ *be the projections associated with some recursive pairing function from* $\mathbb{N} \times \mathbb{N}$ *to* $\mathbb{N}$. *For each type* σ *we define a set* $\mathsf{P}_\sigma^{\mathit{eff}}$ *together with a set* $\mathsf{P}_\sigma^{\omega,\mathit{eff}}$ *of functions* $f : \mathbb{N} \to \mathsf{P}_\sigma^{\mathit{eff}}$ *as follows*:

$$\mathsf{P}_0^{\mathit{eff}} = \mathbb{N}_\perp.$$

$$\mathsf{P}_0^{\omega,\mathit{eff}} \cong \mathbb{N}_{p\,rec}^{\mathbb{N}}.$$

$$\mathsf{P}_{\sigma\to\tau}^{\mathit{eff}} = \{f : \mathsf{P}_\sigma^{\mathit{eff}} \to \mathsf{P}_\tau^{\mathit{eff}} \mid \forall g \in \mathsf{P}_\sigma^{\omega,\mathit{eff}}.\ f \circ g \in \mathsf{P}_\tau^{\omega,\mathit{eff}}\}.$$

$$\mathsf{P}_{\sigma\to\tau}^{\omega,\mathit{eff}} = \{f : \mathbb{N} \to \mathsf{P}_{\sigma\to\tau}^{\mathit{eff}} \mid \forall g \in \mathsf{P}_\sigma^{\omega,\mathit{eff}}.\ \Lambda n.\, f(\mathit{fst}\, n)(g(\mathit{snd}\, n)) \in \mathsf{P}_\tau^{\omega,\mathit{eff}}\}.$$

(This is a mild variation on the definition given in Longo and Moggi [1984b] — the latter relies on a theorem ensuring the existence of suitable pairing operations at higher types.) Inspired by this characterization, Longo and Moggi introduced another cartesian closed category extending ***EN***: the category of *generalized numbered sets* (Longo and Moggi [1984a]). Moggi later clarified the relationship between this category and the recursive topos (Moggi [1988]).

Subsequent work has shown how P and $\mathsf{P}^{\mathit{eff}}$ arise naturally from various realizability models. We will continue this part of the story in Section 5 below.

4.2.4. *PCF and parallelism.* We now return to discuss the important connections between P, $\mathsf{P}^{\mathit{eff}}$ and formal languages, beginning again with Scott's early work. In Scott [1993], Scott showed that P provided a model for a simple formal language inspired directly by Platek's work (Section 4.1.1). In the context of Scott's paper, this served as the term language of a logic intended for reasoning about computable functions, which later became known as LCF.

[16]It is worth noting, in passing, that the type structure over N in this topos does not coincide with any of our type structures of total computable functionals, but rather yields the *generalized Banach-Mazur functionals*: see Mulry [1982] and cf. Section 2.2.

Scott's language for functions may be presented as follows. As usual, we will work with the simple types over $\overline{0}$.[17]

Definition 4.16 (PCF).[18] *Let* (combinatory) PCF *be the language consisting of the well-typed expressions built up via application from the constants*

$$\begin{aligned}
\mathtt{K}_{\sigma\tau} &: \sigma \to \tau \to \sigma \\
\mathtt{S}_{\rho\sigma\tau} &: (\rho \to \sigma \to \tau) \to (\rho \to \sigma) \to (\rho \to \tau) \\
\mathtt{Y}_{\sigma} &: (\sigma \to \sigma) \to \sigma \\
\mathtt{if} &: \overline{0} \to \sigma \to \sigma \\
\mathtt{0} &: \overline{0} \\
\mathtt{succ} &: \overline{0} \to \overline{0} \\
\mathtt{pred} &: \overline{0} \to \overline{0}
\end{aligned}$$

The similarity to Platek's definition of recursive definability (Definition 4.3) is clear. The constant `if` takes over the role of Platek's D; $\mathtt{0}, \mathtt{succ}, \mathtt{pred}$ correspond to a set of basic computable elements; and Platek's I is redundant anyway, since one can define $I = SKK$.

Of course, there are some arbitrary choices involved in this definition, and many mild variants of it lead to essentially the same language. For instance, one might drop the constants K and S, and instead consider the simply typed λ-calculus over the remaining constants (this is in fact the version that is most commonly considered today). Alternatively, one might define PCF as "System T plus general recursion": that is, we simply augment the definition of Gödel's System T (Section 3.1.1) with the constants $\mathtt{Y}_\sigma$. The possibility of such variants was already clear to Scott in Scott [1993]; they are all intertranslatable and for our purposes have the same expressive power, so we may freely refer to any of them as "PCF".

Scott gave an interpretation of PCF in P in the spirit of Platek's definition. The constants $\mathtt{K}, \mathtt{S}, \mathtt{0}, \mathtt{succ}, \mathtt{pred}$ are interpreted in the obvious way. The combinator `if` is interpreted by the element *if* of P is given by

$$\mathit{if}\ (0)(x)(y) = x, \quad \mathit{if}\ (n+1)(x)(y) = y, \quad \mathit{if}\ (\bot)(x)(y) = \bot.$$

The combinator $\mathtt{Y}_\sigma$ is interpreted by the function Y_σ which assigns to any $f \in \mathsf{P}_{\sigma\to\sigma}$ the least fixed point of f in P_σ. (Note that because all functionals in P are continuous, we may construct Y_σ simply by an iteration up to ω — we do not require transfinite iterations as in Section 4.1.1.) Finally, application

[17]Actually, Scott's system also had a ground type of truth-values as well as one of integers, and hence a slightly different selection of constants. Here, as elsewhere, we will content ourselves with representing "true" by zero and "false" by any other natural number. Otherwise, we have kept closely to Scott's original definition.

[18]The name 'PCF' first appeared in Milner [1977].

in PCF is interpreted by application in P. We thus have the notion of a *PCF-definable* element of P.

Scott's stated intention in introducing the above language was to provide "a restricted system that is specially designed for algorithms". However, he presented the system purely as a mathematical language for defining elements of P rather than as an executable "programming language", and initially it was not even immediately obvious whether all the PCF-definable elements of P were effectively computable in any reasonable sense (see Scott [1993, Section 4]).

In fact (as Scott quickly realized), it is easy to show that every PCF-definable element P is effective at least in the sense that it is an element of $\mathsf{P}^{\mathit{eff}}$. However, the converse is not true: the function *por* considered by Platek (Definition 4.4) is effective but not PCF-definable. It is clear that *por* is in some sense computable — one can evaluate $por\,(x)(y)$ by evaluating x and y "in parallel" — but like Platek, Scott noted that this kind of algorithm has a significantly different flavour from the means of computation that suffice to compute the PCF-definable elements (Scott [1993, Section 4]):

> Do we enjoy this new flavor enough to call it computable? Some people would say yes, but I wonder.

It thus began to appear that there might be two reasonable notions of computable element in P: the notion of effective element (given by $\mathsf{P}^{\mathit{eff}}$), and the more restrictive notion of PCF-definable element.

The next major results were obtained independently by Plotkin (Plotkin [1977]) and Sazonov (Sazonov [1976a]). (Closely related results were also obtained around the same time by Feferman (Feferman [1977a]).) These authors showed that the gap between PCF computability and effectivity could be bridged by augmenting PCF with just two operations of a "parallel" flavour:

THEOREM 4.17. *Let* PCF^{++} *be the language* PCF *extended with the two constants*

$$\texttt{por}\colon \overline{0} \to \overline{0} \to \overline{0}, \quad \texttt{exists}\colon (\overline{0} \to \overline{0}) \to \overline{0}.$$

Extend the interpretation of PCF in P *by interpreting* `por` *by the function por* (*see Definition* 4.4), *and* `exists` *by the functional given by*

$$exists(f) = \begin{cases} 0 & \text{if } f(n) = 0 \text{ for some } n \in \mathbb{N}, \\ 1 & \text{if } f(\bot) = k+1 \text{ for some } k \in \mathbb{N} \\ & \quad (\text{hence } f(x) = k+1 \text{ for all } x), \\ \bot & \text{otherwise.} \end{cases}$$

Then the elements of $\mathsf{P}^{\mathit{eff}}$ *are exactly the* PCF^{++}*-definable elements of* P.[19]

[19]In fact Plotkin considered a parallel conditional operator rather than parallel or, but it is easy to show that these are interdefinable over PCF (see Stoughton [1991a]).

Notice that the functional *exists* here is a genuinely effective "parallel search operator", quite different in flavour from the functionals ${}^k\exists$ considered in Section 3.2.2.

As pointed out in Feferman [1977a], this was a significant result in that it showed that this notion of computability on P could be captured by a finite collection of inductive schemata, somewhat in the spirit of Kleene's S1–S9. Interestingly, it seems that Kleene himself never considered parallel operations as a way of getting more expressive power, even though he was specifically interested in the problem of identifying a class of "all" computable functions at higher types.

4.2.5. *Operational semantics.* So far we have considered PCF and PCF^{++} simply as formal languages for defining elements of P. However, a crucial step was made by Plotkin (Plotkin [1977]), who showed how to give an *operational semantics* (that is, a set of symbolic evaluation rules) for these systems, turning them into executable programming languages. This meant that the "meaning" of programs (that is, their evaluation behaviour) could be defined without reference to a model such as P.

We illustrate the idea by giving one possible operational semantics for our version of PCF^{++}. Our definition is in a rather different style from Plotkin's, being closer in spirit to Milner [1977]. Specifically, we will define a transitive relation $\rightsquigarrow$ on terms corresponding to a notion of many-step reduction. We write $\widehat{k}$ as an abbreviation for $(\mathtt{succ}^k\,\mathtt{0})$, and $\bot_\sigma$ for the term $\mathtt{Y}_\sigma\mathtt{I}$ where $\mathtt{I} = \mathtt{SKK}$.

Definition 4.18 (Operational semantics for PCF^{++}). *Let $\rightsquigarrow$ be the smallest transitive relation on terms of* PCF^{++} *satisfying the following clauses, in which we assume all terms are well-typed.*

- $\mathtt{K}UV \rightsquigarrow U$, $\quad \mathtt{S}UVW \rightsquigarrow (UW)(VW)$, $\quad \mathtt{Y}U \rightsquigarrow U(\mathtt{Y}U)$.
- $\mathtt{pred}\,\mathtt{0} \rightsquigarrow \mathtt{0}$, $\quad \mathtt{pred}\,\widehat{k+1} \rightsquigarrow \widehat{k}$.
- $\mathtt{if}\,\mathtt{0}\,U\,V \rightsquigarrow U$, $\quad \mathtt{if}\,\widehat{k+1}\,U\,V \rightsquigarrow V$.
- $\mathtt{por}\,\mathtt{0}\,V \rightsquigarrow \mathtt{0}$, $\quad \mathtt{por}\,V\,\mathtt{0} \rightsquigarrow \mathtt{0}$, $\quad \mathtt{por}\,\widehat{j+1}\,\widehat{k+1} \rightsquigarrow \widehat{1}$.
- *If* $U\,\widehat{n} \rightsquigarrow \mathtt{0}$ *then* $\mathtt{exists}\,U \rightsquigarrow \mathtt{0}$.
- *If* $U\,\bot_\sigma \rightsquigarrow \widehat{k+1}$ *then* $\mathtt{exists}\,U \rightsquigarrow \widehat{1}$.
- *If* $U \rightsquigarrow U'$ *then* $UV \rightsquigarrow U'V$.
- *If* $U \rightsquigarrow U' : \overline{0}$ *then* $c\ U \rightsquigarrow c\ U'$ *for* $c = \mathtt{succ}, \mathtt{pred}, \mathtt{if}, \mathtt{por}, \mathtt{exists}$, *and* $\mathtt{por}\ V\ U \rightsquigarrow \mathtt{por}\ V\ U'$.

We say a term $U : \overline{0}$ evaluates to *some (necessarily unique) natural number* k *if* $U \rightsquigarrow \widehat{k}$. *A term* $U : \overline{0}$ converges *if it evaluates to some* k; *otherwise* U diverges.

Let us write $[\![\, U \,]\!]$ for the element of P denoted by a PCF^{++} term U. The following fundamental result (essentially Theorem 3.1 of Plotkin [1977]) says that the operational and denotational semantics agree.

THEOREM 4.19 (Adequacy of P). *For any closed term $U : \overline{0}$, U evaluates to k iff $[\![U]\!] = k$.*

The proof makes use of powerful ideas developed in Tait [1967] to prove strong normalization for calculi such as System T (see Section 3.1.1).

Plotkin's work on PCF marked something of a break between the older recursion theory tradition and what was to become the modern computer science tradition. Roughly speaking, whereas the former usually treated formal languages or inductive schemata principally as ways of picking out a class of computable elements from some predefined mathematical structure such as S or P, the tendency in computer science has been to take the programming languages as primary and then look for mathematical structures in which they can be interpreted. (One can think of exceptions to this, of course, but some cultural difference along these lines can be discerned in the literature and persists to this day.) The ideas of operational semantics, which allow us to give standalone syntactic definitions of programming languages, are what make the latter approach viable.

We can perhaps illustrate this point of view by using the operational definition of PCF^{++} to give an independent and purely syntactic construction of $\mathsf{P}^{\mathit{eff}}$ (up to isomorphism). For PCF^{++} terms $U, V : \overline{0}$, let us write $U \equiv V$ if U and V either both diverge or both evaluate to the same number k. Let A be the type structure of PCF^{++} terms, and let $\equiv_\sigma$ be the partial equivalence relations induced by the extensional collapse construction with respect to $\equiv$ (Definition 1.2). It is not too hard to show that $U \equiv_\sigma V$ iff $[\![U]\!] = [\![V]\!] \in \mathsf{P}_\sigma$. It follows by Theorem 4.17 that $(A_\sigma / \equiv_\sigma) \cong \mathsf{P}^{\mathit{eff}}_\sigma$; thus, $\mathsf{P}^{\mathit{eff}}$ is isomorphic to $\mathsf{EC}(A, \equiv)$.

4.3. PCF and sequential computability. By ignoring the references to `por` and `exists` in Definition 4.18, we obtain a standalone operational definition of PCF. Using such a definition, Plotkin was able to give rigorous proofs that *por* and *exists* are not PCF-computable, by an analysis of possible reduction behaviours. (A remarkably similar result had been obtained quite independently in Sasso [1971] — see Odifreddi [1989, p. 188].) Despite this incompleteness, the feeling persisted that the notion of computability embodied by PCF was a natural one and of interest in its own right.

Intuitively, computations in PCF proceed in a "sequential" manner in the sense that there is a single thread of computation — we never find two disjoint subterms of a term being evaluated at the same time. Since very many practical programming languages also have this sequential character, PCF appeared attractive from a computer science perspective as a prototypical programming language for suitable theoretical study.

4.3.1. *PCF versus S1–S9.* Before surveying the main body of research relating to PCF, we digress briefly to comment on its relationship to Kleene's S1–S9. It is sometimes remarked that PCF is essentially equivalent to S1–S9, but in our view this idea needs to be treated with caution, as there is a subtle issue here which seems never to have been clearly explained in the published literature.

Even if we restrict attention to the type structure P, taking a literal reading of S1–S9 as given in Kleene [1959b] (or our Definition 3.2), it is true that all the Kleene computable elements of P are PCF-definable, but not *vice versa*. This is because there is a difference between saying that $\{m\}(\boldsymbol{x})$ is not defined (by the inductive definition of the ternary relation $\{m\}(\boldsymbol{x}) = y$) and saying that $\{m\}(\boldsymbol{x}) = \bot$. Indeed, the fact that in P the notion of non-termination has been objectified by $\bot$ means that if we wish to have $\{m\}(\boldsymbol{x}) = \bot$, then this triple must be explicitly generated by the inductive procedure. Now suppose we attempt to construct an index m for, say, the fixed point operator $Y_1 \in \mathsf{P}_{(\overline{1}\to\overline{1})\to\overline{1}}$. By invoking S9 and appealing to Kleene's second recursion theorem, we can certainly obtain an index m with the property that $\{m\}(F) \simeq F(\{m\}(F))$ for all $F \in \mathsf{P}_{\overline{1}\to\overline{1}}$. But now, even if we specialize F to the constant function $\Lambda g.\Lambda x.0$, we cannot conclude that $\{m\}(F) = 0$, since strictly speaking we cannot assign a meaning to $F(\{m\}(F))$ before $\{m\}(F)$ has been given a meaning! This is because F is just an ordinary mathematical function on the set $\mathbb{N} \sqcup \{\bot\}$. All we can deduce is that *if* $\{m\}(F)$ *means anything* (whether $\bot$ or a number) then $F(\{m\}(F)) = 0$ and so $\{m\}(F) = 0$. In fact, it can be shown that the element Y_1 is not Kleene computable under this strict interpretation.

One can overcome this problem by reinterpreting S1–S9 not as the inductive definition of a set of triples $(m, \boldsymbol{x}, y)$, but as the recursive definition of a total function $\{-\}(-) \colon \mathbb{N} \times X(\mathsf{P}) \to \mathbb{N}_\bot$ obtained as a gigantic simultaneous least fixed point. (Alternatively, one could perhaps recast it as the definition of a relation $\{m\}(\boldsymbol{x}) \sqsupseteq y$). It appears that this is often what people have in mind when referring to Kleene computability over P, and it *is* true that this more generous notion coincides with PCF-definability.

Even so, it seems to us questionable whether this latter interpretation is true to the *spirit* of S1–S9 as manifested in Kleene's papers. As remarked earlier, the schemata S1–S9 were introduced by Kleene to capture the ideal of computing with functions as pure extensions, whose characteristic behaviour is that when presented with a specified object of lower type they simply return an answer (or perhaps diverge). It was a conscious shift in perspective on Kleene's part to regard computations as acting on more intensional representations of functions, such as the formal expressions in Kleene [1978] or the oracles in Kleene [1980]; and for this purpose Kleene introduced S11. Now the strategy required to compute a functional such as Y_1 makes essential use of the idea that an object F of type $\overline{1} \to \overline{1}$ can not only provide answers when presented with specified arguments, but also respond with questions of its own when presented with unspecified (or incompletely specified) arguments. And if F can do this, it is (in our view) behaving as something more than a pure extension. It therefore seems to us more accurate, both in letter and in spirit, to say that PCF essentially corresponds to Kleene's S1–S8 plus S11 (this is indeed stated explicitly in Nickau [1994]). It is worth remarking that Kleene himself never used S9 in connection with hereditarily partial functionals.

One can, however, define a weaker language than PCF that does correspond in expressive power to the strict interpretation of S1–S9, essentially by using S1–S9 themselves as the basis of a language for *partial* functionals. The notion of computability embodied by this language seems to be a very natural one and deserves more attention than it has so far received. Such a language captures an intuitively appealing idea of "computing with pure extensions", and moreover seems to offer the most natural route to the study of Kleene computability even in total settings. We will say more about this notion of computability in Part II, where we will introduce some associated type structures K and $\mathsf{K}^{\mathit{eff}}$.

Even weaker languages have also been considered. For instance, one can consider Gödel's System T extended with the minimization operator μ of ordinary recursion theory. Some preliminary expressivity results on this and related systems have recently been considered by Berger (Berger [2000]). Closely related issues have also recently been considered in Niggl [1999] and Normann and Rørdam [2002].

4.3.2. *The full abstraction problem.* The operational semantics of PCF gives rise to a notion of *observational equivalence* of PCF programs.

DEFINITION 4.20. *We say the PCF terms $U, V : \sigma$ are* observationally equivalent (*and write $U \approx_\sigma V$*) *if, for all term contexts $C[-]$ such that $C[U]$, $C[V]$ are terms of type $\overline{0}$, $C[U]$ evaluates to k iff $C[V]$ evaluates to k.*

This is a natural notion from a programming point of view, since $U \approx V$ means that it is always safe to replace U by V in any larger program without affecting the overall result. It is therefore natural to ask whether one can give a mathematical characterization of observational equivalence (in PCF or in other languages).

An important result of Milner (Milner [1977]) shows that two PCF terms are observationally equivalent iff, intuitively, they induce the same functions on terms of lower type:

THEOREM 4.21 (Context Lemma). *For closed PCF terms*

$$U, V : \sigma_1 \to \cdots \to \sigma_r \to \overline{0},$$

we have $U \approx V$ iff, for all closed $W_1 : \sigma_1, \dots, W_r : \sigma_r$, $UW_1 \dots W_r$ evaluates to k iff $VW_1 \dots W_r$ does.

The idea that the behaviour of PCF terms is completely determined by the functions they induce is expressed by saying that PCF is a *purely functional* programming language.

We may now see how to define a standalone type structure of PCF-definable functionals from the definition of PCF itself. This is analogous to the construction of $\mathsf{P}^{\mathit{eff}}$ from PCF^{++} mentioned at the end of Section 4.2.5.

Definition 4.22 (PCF type structure). *Let B be the type structure consisting of PCF terms, and let $\mathsf{Q}^{\mathit{eff}}$ be the extensional collapse of B with respect to $\approx_0$.*

It follows easily from the context lemma that $\mathsf{Q}^{\mathit{eff}}$ is actually the *quotient* of B modulo observational equivalence. We may regard the type structure $\mathsf{Q}^{\mathit{eff}}$ as embodying the notion of "sequentially computable functional" represented by PCF.

It is natural to ask if one can give a more mathematically illuminating description of $\mathsf{Q}^{\mathit{eff}}$ than the syntactic definition given above. In particular, can we find a mathematically natural type structure Q within which $\mathsf{Q}^{\mathit{eff}}$ sits as an "effective substructure", in the way in which $\mathsf{P}^{\mathit{eff}}$ is a substructure of P? More or less equivalently, can we give a denotational interpretation of PCF such that $[\![\, U \,]\!] = [\![\, V \,]\!]$ iff $U \approx V$? In computer science terminology, such an interpretation would be called *fully abstract*.

Note that the interpretation of PCF in P does not fulfil this requirement. It follows from the adequacy theorem that if $[\![\, U \,]\!] = [\![\, V \,]\!]$ in P then $U \approx V$, but the converse fails. This is because one can find terms $U, V : \sigma \to \tau$ such that $[\![\, U \,]\!](x) = [\![\, V \,]\!](x)$ for all PCF-definable elements $x \in \mathsf{P}_\sigma$, but $[\![\, U \,]\!](x) \neq [\![\, V \,]\!](x)$ for some non-definable elements such as *por* (see Plotkin [1977]).

The problem of trying to give a good mathematical characterization of the "sequential" functionals at higher types, and in particular of finding a suitable type structure Q with the above properties, became known as the *full abstraction problem* for PCF, and was to receive much attention in theoretical computer science. The early investigators had in mind a denotational model consisting of CPOs of some kind, though chain completeness is not an essential requirement. Plotkin and Milner (Milner [1977]) showed for a large class of potential CPO models of PCF that full abstraction holds iff all the finite elements are PCF-definable. As shown by Milner, it follows that there is up to isomorphism just one (order-extensional) fully abstract CPO model. However, Milner's construction of this model was still syntactic in flavour and was not felt to yield a good mathematical characterization of sequentiality.

Characterizations of sequentiality were obtained easily enough for first order functions from $\mathbb{N}_\perp^r$ to $\mathbb{N}_\perp$. (By a mild abuse of notation we will write $\mathbb{N}_\perp^0 \to \mathbb{N}_\perp$ to mean $\mathbb{N}_\perp$, and $\mathbb{N}_\perp^{r+1} \to \mathbb{N}_\perp$ to mean $\mathbb{N}_\perp \to (\mathbb{N}_\perp^r \to \mathbb{N}_\perp)$.) The following definition (by induction on r) was given by Milner:

Definition 4.23. *A monotone function $f : \mathbb{N}_\perp^r \to \mathbb{N}_\perp$ is* sequential *if either it is constant, or there is some i such that for all $x_i \in \mathbb{N}_\perp$ the function*

$$\Lambda x_1 \dots x_{i-1} x_{i+1} \dots x_r .\, f(x_1)\dots(x_i)\dots(x_r) : \mathbb{N}_\perp^{r-1} \to \mathbb{N}_\perp$$

is sequential.

Intuitively, f is sequential if at each stage in the computation of $f x_1 \dots x_r$, either f can return an answer, or else it cannot because it needs to know the value of some argument x_i. (If there is only one such i, this is the argument that needs to be evaluated next in the computation; if there is more than one, we have some choice in which argument to evaluate next.) Similar (though not identical) ideas were present in Kleene's definition of unimonotonicity for type 2 oracles (Section 4.1.2). A different but equivalent characterization was also obtained independently by Vuillemin (Vuillemin [1973]).

It is easily shown that a *finite* element of $\mathsf{P}_{\overline{0}^r \to \overline{0}}$ is sequential iff it is PCF-definable, so that a model for PCF that included only sequential functions at first order types would be fully abstract for types of level 2. However, a counterexample due to Trakhtenbrot (Trakhtenbrot [1975]; see also Sazonov [1976a]) shows that not every *effective* sequential element of $\mathsf{P}_{\overline{0}^r \to \overline{0}}$ is PCF-definable — it can happen that at some stage some appropriate choice of index i exists, but there is no effective way to compute it.

The problem of characterizing sequentiality at higher types turned out to be much more difficult, and was the focus of a great deal of research effort in theoretical computer science. Milner (Milner [1977]) and later Mulmuley (Mulmuley [1987]) gave constructions of the fully abstract CPO model, but both of these referred in some way to the operational semantics of PCF and so did not qualify as an independent mathematical characterization. Several other models for PCF were constructed: Berry's *stable* and *bistable* models (Berry [1978]); the Berry-Curien *sequential algorithms* model (Berry and Curien [1982], Curien [1993]) based on Kahn and Plotkin's notion of *concrete data structure* (Kahn and Plotkin [1993]); and the Bucciarelli-Ehrhard *strongly stable* model (Bucciarelli and Ehrhard [1991b], Ehrhard [1993]). For a detailed survey of this material and further references, we recommend Ong [1995]. This line of research gave much insight into the difficulty of the full abstraction problem, and generated many counterexamples illustrating the subtlety of the notion of observational equivalence. We will not describe the above models in detail here, since as far as PCF is concerned the main point about them is that they are *not* fully abstract. However, both the sequential algorithms model and the strongly stable model turned out to be important in connection with other notions of computability, and we will return to them below.

Given that the only known constructions of a fully abstract model were felt to be unsatisfactory, it was natural to ask what exactly were the criteria for a "good" solution to the full abstraction problem. One possible criterion was proposed in Jung and Stoughton [1993]: when restricted to *finitary* PCF (that is, the fragment of PCF generated by the ground type of booleans), the model construction should be *effective*. In other words, given a simple type σ over the booleans it should be possible to effectively compute a complete description of the (finite) semantic object representing σ. This criterion rules out syntactic constructions such as Milner's; it admits the other models mentioned above, though they are not fully abstract.

If an extensional fully abstract model satisfying the Jung-Stoughton criterion existed, it would follow that observational equivalence for finitary PCF was decidable. However, around 1996 Loader showed:

THEOREM 4.24. (*Loader* [2001]) *Observational equivalence in finitary PCF is undecidable.*

The proof involves a tricky encoding of semi-Thue problems. Loader's result represents one of the most important advances in our understanding of PCF: it closes the door on a large class of attempts at the full abstraction problem, and indeed shows that it was not possible to give any good finitary analysis of PCF-sequentiality considered purely as a property of functions. (Remarkably, however, the fully abstract model for unary PCF — that is, PCF over a ground type with a single terminating value $\top$ — *is* effectively presentable; see Loader [1998].)

4.3.3. *Intensional semantics for PCF.* Another possible approach is to seek a good mathematical description of the *algorithms* embodied by PCF terms rather than of the functions they compute. This approach does in fact lead to good models of sequential computation, albeit of an "intensional" nature. One can regard such models as occupying a kind of middle ground between operational and denotational semantics as traditionally conceived: elements of the model are typically computation strategies with a dynamic operational behaviour, but many of the inessential details present in an operational definition of a language are abstracted out.

An approach of this kind was first successfully carried through by Sazonov (Sazonov [1975], [1976b], [1976c]), whose work was explicitly formulated in terms of Scott's LCF but was independent of the work of Milner and Plotkin, and indeed remained little known in the West until a brief description of it appeared in Hyland and Ong [2000]. In effect, Sazonov gave a model of PCF-style sequential computation in terms of Turing machines with oracles, much in the same spirit as Kleene's characterization of S1–S9 computations in Kleene [1962c], [1962d]. A Turing machine for a function of type $\overline{n}$ communicates with its argument (an oracle of type $\overline{n-1}$) by feeding it with a description of a Turing machine for type $\overline{n-2}$. As in Kleene [1962c], computations have the character that functionals of type $\overline{n-k}$ are represented by (codes for) Turing machines when k is even, and by pure oracles when k is odd. At the heart of Sazonov's work is his notion of the strategy followed by a Turing machine: sequentiality is enforced by a requirement that when a machine invokes an oracle, it must receive an answer before it can continue. Although Sazonov's formalization is somewhat complicated and rather dependent on the use of explicit codings for Turing machines, this approach succeeds in capturing the notion of PCF computability at all finite types and very closely anticipates many features of later models. (In fact, it could be argued that Sazonov had already accomplished what Kleene was trying to achieve in the papers from Kleene [1978] onwards!) An account of Sazonov's work framed in more modern computer science terms is given in Sazonov [1998].

A somewhat similar approach was pursued for many years by Gandy and his student Pani, who however concentrated more on the problem of characterizing the PCF-definable elements of P. This approach was apparently influenced by Kleene [1978], and emphasized the idea of computations as dialogues between two participants. The information available to the participants at each stage is represented by finite elements of P (or similar entities). Gandy's insights had a significant influence on the computer science community, though unfortunately Gandy never completed a written account of his ideas and their exact form remains unclear.

Within computer science itself, an abstract formulation of the essence of PCF sequentiality in terms of dialogue games was achieved around 1993, when three closely related models of PCF were obtained (simultaneously and more or less independently) in Abramsky, Jagadeesan and Malacaria (Abramsky, Jagadeesan, and Malacaria [2000]), Hyland and Ong (Hyland and Ong [2000]), and Nickau (Nickau [1994]). Again, the formal details can appear rather complicated, but the essential ideas can be easily grasped via an example. Using λ-calculus notation and some obvious abbreviations here for convenience, suppose we are given the PCF terms

$$\begin{aligned} F &\equiv \lambda gh.\,(g\ 3) + (g(\mathrm{Y}h)) && : \overline{1} \to \overline{1} \to \overline{0} \\ g &\equiv \lambda x.\, x + 2 && : \overline{1} \\ h &\equiv \lambda x.\, 4 && : \overline{1} \end{aligned}$$

and we wish to evaluate Fgh. The computation can be modelled as a dialogue between a Player P, who follows a strategy determined by F, and an Opponent O, whose strategy is determined by g and h. Moves in the game are either questions (written '?'), or answers (natural numbers) which are matched to previous questions. Furthermore, the moves are associated with occurrences of $\overline{0}$ in the type in question, according to their "meaning" in the context of the game. The game always starts with a question by Opponent, who asks for the final result of the computation. We show in Figure 2 the dialogue corresponding to the computation of Fgh, along with an informal paraphrase of the meaning of each move. (In fact, the order of moves here precisely matches the order of interactions between Turing machines and oracles in Sazonov's approach.)

FIGURE 2. A game play corresponding to a PCF computation

	$(\overline{0} \to$	$\overline{0}) \to$	$(\overline{0} \to$	$\overline{0}) \to$	$\overline{0}$	
O:					?	What is $F(g)(h)$?
P:		?				What is $g(x)$?
O:	?					What is x here?
P:	3					$x = 3$.
O:		5				In that case, $g(x) = 5$.
P:		?				All right. Now what is $g(x')$?
O:	?					What is x' here?
P:				?		What is $h(y)$?
O:				4		$h(y) = 4$ (whatever y is).
P:	4					Then $x' = 4$.
O:		6				In that case, $g(x') = 6$.
P:					11	Well then, $F(g)(h) = 11$.

In the game model of Hyland and Ong [2000], for instance, two main constraints on strategies are imposed: *well-bracketing* (every answer must match the most recent pending question) and *innocence* (roughly, the participants must decide on their moves purely on the basis of the answers received to previous questions, not on how these answers were obtained). Another feature of this model is that every move apart from the first one is explicitly *justified* by some previous move: answers are justified by the questions they answer, and questions are justified by earlier questions which open up the appropriate part of the game.

All of the game models mentioned yield definability results for PCF: every recursive strategy is the denotation of some PCF term. In fact, they all give rise to the same extensional type structure Q, which satisfies the criteria mentioned following Definition 4.22, and which seems to be the natural candidate for such a type structure. (It is, however, an open problem whether the Q_σ are all CPOs.) Note that the elements of Q are in general infinite equivalence classes of strategies, so these constructions do not have quite the finitary character that one might have liked (this is inevitable in view of Theorem 4.24). However, this does not mean that no useful analysis has been achieved. In our view, the main insight offered by the perspective of game semantics is the idea that a *parity* (O or P) may be consistently assigned to the implicit interactions between the subterms of a PCF term. This parity is extra structure present in the game models — it is not present in the raw operational definition of PCF. The game-theoretic analysis seems to us to be deep enough to resolve some interesting and purely syntactic questions about PCF, though significant results of this nature have yet to be worked out in detail.

The ideas of game semantics have also been successfully applied to yield characterizations of non-extensional notions of computability — see Section 6.2.

4.3.4. *Other work.* We now mention a few miscellaneous topics to conclude our survey of what is known about PCF computability.

A remarkable result of Sieber (Sieber [1992]) gives a complete characterization of the PCF-definable finite elements at types of level 2 as those that are invariant under a class of logical relations known as sequentiality relations. The idea of using invariance under logical relations to construct a model was carried much further by O'Hearn and Riecke (O'Hearn and Riecke [1995]) who obtained a fully abstract model in this way; however, their result depends only on general facts about λ-definability, and so it is unclear how much it reveals about sequentiality as such.

Some attention has also been given to other variants of PCF. The version considered above is the original call-by-name one, but one can equally well define a version of PCF with a call-by-value evaluation strategy, which can be naturally interpreted in CBV type structures (the idea essentially appears in Plotkin [1983, Chapter 3]). There is also a third possibility: the *lazy PCF* of Bloom and Riecke [1989]. The relationships between these languages were investigated in Sieber [1990], Riecke [1993], Longley [1995, Chapter 6]. The picture that has emerged is that these languages are sufficiently interencodable that, from a denotational perspective at least, they can all be regarded as embodying the same abstract "notion of computability". We will explain this point of view in more detail in Part II.

Another area of recent interest has been the relationship between PCF computability and the notions of *total* functional considered in Sections 3.3 and 3.4. The general idea here is to ask what total functionals can be computed in PCF, though this question can be made precise in several different ways. For example, we have already seen that C arises as an extensional collapse of P (Theorem 4.13); one can therefore ask which elements of C are represented by PCF-definable elements of P. It is not hard to see that all Kleene computable elements of C are PCF-definable in this way. More surprising is the fact that the fan functional Φ (see Section 3.3.1) is PCF-definable by means of a clever higher type recursion. This fact appeared in Berger's thesis (Berger [1990]), and was also independently known to Gandy. The following more general result was conjectured in Cook [1990] and Berger [1993], and proved by Normann in Normann [2000].

Theorem 4.25. *The PCF-definable elements of* C *are exactly those in* RC.

Equivalently (in the light of Theorems 4.13 and 4.17), the PCF-definable elements of C coincide exactly with the PCF^{++}-definable ones. Normann's proof is both ingenious and beautiful and is one of the highlights of the subject. Around the same time, Plotkin (Plotkin [1997]) considered other possible notions of totality in PCF (such as that given by an extensional collapse of the term model for PCF itself) and investigated the differences between the various notions.

Finally, a few papers have been devoted to the degree theory induced by the notion of relative PCF-definability. An element $x \in \mathsf{P}^{\mathit{eff}}$ (for example) is PCF-definable relative to $y \in \mathsf{P}^{\mathit{eff}}$ if there is a PCF-definable element f such that $f(y) = x$; the corresponding equivalence classes are known as *degrees of parallelism*. The lattice of degrees of parallelism was introduced by Sazonov (Sazonov [1976a]); like other lattices of degrees, its structure would appear to be extremely complicated. Sazonov mentioned several examples of distinct degrees and some relationships between them: for instance, the functions *por* and *exists* represent incomparable degrees. A few other results in a similar spirit appear in Trakhtenbrot [1975]. Bucciarelli (Bucciarelli [1995]) undertook a somewhat more systematic study of degrees of parallelism for first order functions; this line of investigation was pursued further by Lichtenthäler (Lichtenthäler [1996]). Degrees of parallelism can be seen as representing notions of computability intermediate between PCF and PCF^{++}; however, to date none of these intermediate notions have established themselves as being of independent mathematical interest.

4.4. The sequentially realizable functionals. Until the mid 1990s, it seemed reasonable to suppose that the only two respectable notions of hereditarily partial computable functional were those embodied by PCF and PCF^{++} — these were typically referred to as "sequential" and "parallel" computability. However, it emerged more recently that there is another good class of computable functionals which can reasonably be seen as embodying an alternative notion of "sequential" computability, more generous than the PCF one.

The basic idea can be given by a simple example. Let $\mathbb{M}_{rec}$ be the set of monotone computable functions from $\mathbb{N}_\perp$ to $\mathbb{N}_\perp$ (that is, $\mathbb{M}_{rec} = \mathsf{P}_1$), and consider the function $F : \mathbb{M}_{rec} \to \mathbb{N}_\perp$ defined by

$$F(g) = \begin{cases} 0 & \text{if } g(\perp) \in \mathbb{N} \ (\textit{i.e.}, \text{ if } g = \Lambda x.k \text{ for some } k \in \mathbb{N}), \\ 1 & \text{if } g(\perp) = \perp \text{ but } g(0) \in \mathbb{N}, \\ \perp & \text{otherwise } (\textit{i.e.}, \text{ if } g(0) = \perp). \end{cases}$$

Intuitively, the function F can be computed via the following strategy: given a function g, feed it the object $0 \in \mathbb{N}_\perp$ (that is, a program which when run will terminate giving the value 0), and then watch g closely to see whether it ever "looks at" its argument (that is, whether the above program is in fact ever run). If g returns a result without looking at its argument, we return 0; if g returns a result having looked at the argument, we return 1.

This is a perfectly effective and intuitively "sequential" way of computing F, as long as g is presented to us in some form of which it is sensible to ask whether it ever looks at its argument. The computation of F thus has to operate on some kind of algorithm or intensional representation for g rather than on its pure extension, although the *result* $F(g)$ is completely determined by the extension g. (As argued in Section 4.3.1, the same is true for PCF computations — the only difference is that here we allow some additional ways of manipulating the intensional representations.) It is easy to see that the function F cannot be defined in PCF: if $g_1 = \Lambda x.\mathit{if}\,(x)(0)(0)$ and $g_2 = \Lambda x.0$, then $g_1 \sqsubseteq g_2$ in the pointwise order but $F(g_1) \not\sqsubseteq F(g_2)$; thus F does not exist even in P since it is not (pointwise) monotone.

It may appear puzzling that a non-monotone function can be considered computable in some sense. The explanation hinges on the fact that computations here operate on intensional objects (as is the case with many of the definitions we have seen), and at this level, computable operations are indeed monotone. Thus, although $g_1 \sqsubseteq g_2$ extensionally, the *algorithm* that computes g_2 is not obtained by extending the algorithm that computes g_1. It is also worth noting that even at the extensional level, functions like F are monotone with respect to a different ordering known as the *stable* order. (Curiously, Kleene came across the possibility of algorithms such as the one described above for F — see Kleene [1985, §13.3] — but decided to rule them out by an extrinsic monotonicity requirement.)

It turns out that there is a mathematically natural type structure containing F and "all things like it", which we shall denote by R. This first appeared in the literature as the type structure arising from the *strongly stable* model of Bucciarelli and Ehrhard [1991b], a domain-theoretic model of PCF intended to capture certain aspects of sequential functionals, and conceived partly as a line of attack on the PCF full abstraction problem. The objects in this model are *dI-domains with coherence*, which are certain CPOs equipped with a class of finite subsets designated as *coherent sets*. The morphisms (equivalently the elements of function spaces) are the *strongly stable functions*, the continuous functions between such CPO which preserve coherent sets and least upper bounds of coherent sets. Though rather complicated to formulate, the construction of this model is as finitary and effective as could be desired, so that (for instance) equality of finite elements is decidable. Ehrhard later gave a simplified presentation in terms of *hypercoherences* (Ehrhard [1993]), and a slightly more abstract analysis of the relevant structure was given in Bucciarelli and Ehrhard [1993].

Another interesting characterization, given by Colson and Ehrhard in Colson and Ehrhard [1994], showed that R in some sense arises naturally from the class of (infinitary) first order sequential functions $\mathbb{N}_p^{\mathbb{N}} \to \mathbb{N}_\perp$. (For functions of this type, there is evidently only one reasonable notion of sequentiality. It can be characterized by a simple infinitary generalization of Definition 4.23, or equivalently in the style of Definition 4.27 below.) Note the analogy with Definition 4.15.

Definition 4.26. *For each type σ we define a set R_σ, and a set R^ω_σ of functions from $\mathbb{N}^\mathbb{N}_p$ to R_σ, as follows*:

- $\mathsf{R}_0 = \mathbb{N}_\perp$.
- R^ω_0 *is the set of sequential continuous functions* $\mathbb{N}^\mathbb{N}_p \to \mathbb{N}_\perp$.
- $\mathsf{R}_{\sigma\to\tau}$ *is the set of all functions* $f : \mathsf{R}_\sigma \to \mathsf{R}_\tau$ *such that for all* $g \in \mathsf{R}^\omega_\sigma$ *we have* $f \circ g \in \mathsf{R}^\omega_\tau$.
- $\mathsf{R}^\omega_{\sigma\to\tau}$ *is the set of all functions* $f : \mathbb{N}^\mathbb{N}_p \to \mathsf{R}_{\sigma\to\tau}$ *such that for all* $g \in \mathsf{R}^\omega_\sigma$ *the function* $\Lambda r : \mathbb{N}^\mathbb{N}_p.\, f(\mathit{fst}\ r)(g(\mathit{snd}\ r))$ *is in* R^ω_τ.

The above characterizations say nothing about whether the elements of R that are not definable in PCF are sequentially computable in any reasonable sense. However, Ehrhard showed in Ehrhard [1996] that there is indeed a computational aspect: every element of R can be in some sense computed by a Berry-Curien sequential algorithm. This line of investigation was continued in Ehrhard [1999], where it was shown that R is in fact the extensional collapse of the sequential algorithms model.

A somewhat simpler characterization in the same vein was discovered independently by van Oosten (Oosten [1999]) and Longley (Longley [2002a]), who constructed R from a certain combinatory algebra $\mathcal{B}$. The definition of $\mathcal{B}$ hinges on the observation that a sequential algorithm for computing a function $F : \mathbb{N}^\mathbb{N}_p \to \mathbb{N}_\perp$ (at a given element $g \in \mathbb{N}^\mathbb{N}_p$) can be represented by an infinitely branching *decision tree*, in which each internal node is labelled with a "question" $?n$, (meaning "what is the value of $g(n)$?"), and each leaf either has an undefined label or is labelled with an "answer" $!n$ (meaning "the value of $F(g)$ is n"). Furthermore, such a decision tree can itself be easily coded by an element f of $\mathbb{N}^\mathbb{N}_p$. Likewise, a sequential algorithm of type $\mathbb{N}^\mathbb{N}_p \to \mathbb{N}^\mathbb{N}_p$ can be represented by an infinite forest of such trees, which can again be coded by an element of $\mathbb{N}^\mathbb{N}_p$. All this is very similar to the idea behind Kleene's *associates* (Section 3.3.1).

In the following definition we take $!n = 2n$ and $?n = 2n + 1$.

Definition 4.27 (Van Oosten algebra).

(i) *Let* $\mathit{play} : \mathbb{N}^\mathbb{N}_p \times \mathbb{N}^\mathbb{N}_p \times \mathrm{Seq}(\mathbb{N}) \to \mathbb{N}_\perp$ *be the smallest partial function such that, for all* $f, g \in \mathbb{N}^\mathbb{N}_p$, $\alpha \in \mathrm{Seq}(\mathbb{N})$ *and* $n, m \in \mathbb{N}$,
   - *if* $f\langle\alpha\rangle = !n$ *then* $\mathit{play}(f, g, \alpha) = n$,
   - *if* $f\langle\alpha\rangle = ?n$ *and* $g(n) = m$ *then* $\mathit{play}(f, g, \alpha) = \mathit{play}(f, g, (\alpha; m))$.

(ii) *For* $f \in \mathbb{N}^\mathbb{N}_p$ *and* $n \in \mathbb{N}$, *write* f_n *for the least function such that* $f_n\langle\alpha\rangle = f\langle n; \alpha\rangle$ *for all* α. *Let* $|, \bullet$ *be the operations defined by*

$$f \mid g = \mathit{play}(f, g, [\,]), \qquad f \bullet g = \Lambda n.(f_n \mid g)$$

and let $\mathcal{B}$ *be the applicative structure* $(\mathbb{N}^\mathbb{N}_p, \bullet)$.

The construction of R from $\mathcal{B}$ is now a standard extensional collapse (see also Section 5 below). Define partial equivalence relations $\sim_\sigma$ on $\mathcal{B}$ by

$$f \sim_0 g \text{ iff } f(0) \simeq g(0) \quad \text{(for example)},$$
$$f \sim_{\sigma\to\tau} g \text{ iff } \forall x, y \in \mathcal{B}.\ x \sim_\sigma y \Rightarrow f \bullet x \sim_\tau g \bullet y$$

We may then define R by taking $\mathsf{R}_\sigma = \mathcal{B}/\sim_\sigma$, with application operations induced by $\bullet$.

This construction shows that all the elements of R can be computed or "realized" by sequential algorithms in some sense. In fact, the relationship between $\mathcal{B}$ and the Berry-Curien sequential algorithms model is very close: the relevant objects in the Berry-Curien category can all be obtained as retracts of $\mathcal{B}$ by standard categorical techniques. This and other results connecting up the known characterizations of R appeared in Longley [2002a], where the elements of R were called the *sequentially realizable* functionals. Some further characterizations of R have recently been given in Hyland and Schalk [2002] and Laird [2002].

Longley also explicitly considered the effective analogue $\mathsf{R}^{\mathit{eff}}$ (which may be constructed either as a standalone type structure or as a substructure of R), and argued that this embodied a natural and compelling notion of sequential computability at higher types. One of the main results of Longley [2002a] was that in both R and $\mathsf{R}^{\mathit{eff}}$ the type $\overline{2}$ is *universal* in the same sense in which $\mathbb{T}^\omega$ is universal in P and $\mathsf{P}^{\mathit{eff}}$ (see Theorem 4.14). A closely related fact is the existence of a type 3 functional $H \in \mathsf{R}^{\mathit{eff}}$ such that every element of $\mathsf{R}^{\mathit{eff}}$ is PCF-definable relative to H; indeed, one can define a programming language PCF $+\mathtt{H}$ with an effective operational semantics whose term model provides an alternative characterization of $\mathsf{R}^{\mathit{eff}}$. (The operation $\mathtt{H}$ can in fact be implemented in existing higher order programming languages such as Standard ML; cf. Longley [1999c].)

As we have seen, the functional $F \in \mathsf{R}^{\mathit{eff}}$ mentioned above is not present in $\mathsf{P}^{\mathit{eff}}$. On the other hand, the function $\mathit{por} \in \mathsf{P}^{\mathit{eff}}$ is not present in $\mathsf{R}^{\mathit{eff}}$. We therefore have two incomparable notions of partial computable functional — this is somewhat analogous to the situation for the total type structures RC and HRC as described in Section 3.4. (See also the diagrams in Appendix A). Indeed, one can also make precise an "anti-Church's thesis" in the partial setting, to the effect that there is no possible type structure of computable functionals over $\mathbb{N}_\perp$ that subsumes both $\mathsf{P}^{\mathit{eff}}$ and $\mathsf{R}^{\mathit{eff}}$. One version of such a result was given in Longley [2002a, Section 11]; a slightly stronger version will be presented in Part II.

The type structures $\mathsf{P}^{\mathit{eff}}$ and $\mathsf{R}^{\mathit{eff}}$ both "contain" the type structure $\mathsf{Q}^{\mathit{eff}}$ of Definition 4.22 in some sense (see also Definition A.1). As a curiosity, however, it is worth noting that there are functionals present in both $\mathsf{P}^{\mathit{eff}}$ and $\mathsf{R}^{\mathit{eff}}$ that are not present in $\mathsf{Q}^{\mathit{eff}}$ (see Longley [2002a, Section 11.2], or else

Bucciarelli and Ehrhard [1994], [1991a] where a kind of "intersection" of P and R is constructed).

In conclusion we remark on a few points of comparison between the PCF-sequential and sequentially realizable functionals (see Longley [2002a, Section 12] for a more detailed discussion). The evidence that R and $\mathsf{R}^{\mathit{eff}}$ are natural mathematical objects seems to us very compelling — indeed, at present we possess a much wider range of *prima facie* independent characterizations for R than for Q. Related to this is the fact that R appears to enjoy better structural properties than Q: for instance, the finitary analogue of R is effectively presentable (in contrast to Theorem 4.24 for Q); and R (unlike Q) possesses a universal type. However, one difficulty with $\mathsf{R}^{\mathit{eff}}$ from a practical point of view is that the universal functional H has a high inherent computational complexity (see Royer [2000]); this makes it unlikely that $\mathsf{R}^{\mathit{eff}}$ will ever become the staple notion of sequentially computable functional employed by higher order programming languages.

§5. Realizability models.

§5. Realizability models. We end our discussion of computable functionals by surveying some ideas from the study of *realizability models* that cross-cut several of the topics we have mentioned so far.

The concept of realizability was introduced by Kleene in Kleene [1945], who showed that the notion of recursive function could be used to give a constructive interpretation of arithmetic. Thereafter, many other kinds of realizability were introduced to give constructive interpretations of various logical systems, and hence establish metamathematical results (see Troelstra [1973]). Later, a more model-theoretic perspective emerged (Hyland [1982]), which made it clear that realizability could provide a common semantic setting for both logics and programming languages (such as typed λ-calculi). A survey of the history of realizability has recently been given in Oosten [2002].

The number and variety of notions of realizability that have been studied is very large (see *e.g.*, Oosten [1991] or Hyland [2002] for an overview). Here we shall give the definitions only for a class of models based on *standard* realizability, a simple generalization of Kleene's original notion — these are the models which have received the greatest attention so far. However, we will also allude briefly to realizability models of other kinds.

We start with the following definition, which was (essentially) introduced in Feferman [1975]:

DEFINITION 5.1 (PCAs). *A* partial combinatory algebra (PCA) *is a set A equipped with a partial "application" operation $\cdot : A \times A \rightharpoonup A$, in which there exist elements $k, s \in A$ such that for all $x, y, z \in A$ we have*

$$k \cdot x \cdot y = x, \quad s \cdot x \cdot y\downarrow, \quad s \cdot x \cdot y \cdot z \simeq (x \cdot z) \cdot (y \cdot z).$$

The definition of PCA leads to a rich computational structure: for instance, in any PCA A one can represent the natural numbers by means of an encoding due to Curry, and all recursive functions are then representable by elements of A. We may therefore think of a PCA as an untyped universe of computation in some abstract sense; this seems a reasonable point of view since many naturally arising PCAs are indeed "effective" in nature. Perhaps the leading example of a PCA is *Kleene's first model* K_1, consisting of the natural numbers with Kleene application: $m \cdot n \simeq \phi_m(n)$. Other interesting examples will be mentioned below.

The models we shall consider here are defined as follows:

Definition 5.2 (PERs).

(i) *Given a PCA* $(A, \cdot)$, *a* partial equivalence relation (or PER) *on* A *is a symmetric, transitive binary relation* R *on* A, *i.e., an equivalence relation on a subset of* A. *We write* A/R *for the set of equivalence classes for* R.

(ii) *If* R, S *are PERs on* A, *the PER* S^R *is defined by*

$$S^R(a, a') \text{ iff } \forall b, b' \in A.\ R(b, b') \Rightarrow S(a \cdot b, a' \cdot b').$$

A morphism $f : R \to S$ *is an element of* $A/(S^R)$; *we say* a realizes f *if* $a \in f$. *We write* $\boldsymbol{PER}(A)$ *for the category of PERs on* A *and morphisms between them.*

It is easy to show that $\boldsymbol{PER}(A)$ is a cartesian closed category, with exponentials as suggested by the above definition. In fact, the categories $\boldsymbol{PER}(A)$ turn out to have an extremely rich structure and to offer a common semantic framework for type theories, constructive logics and programming languages.

Girard originally introduced the category $\boldsymbol{PER}(K_1)$ in order to give a semantics for his second order polymorphic λ-calculus, or "System F" (Girard [1972]). Later, $\boldsymbol{PER}(K_1)$ was identified as an important subcategory of Hyland's *effective topos* (Hyland [1982]), a categorical model for higher order logic based on Kleene's realizability interpretation of arithmetic (Kleene [1945]). More generally, $\boldsymbol{PER}(A)$ arises as a subcategory of the *standard realizability topos* $\boldsymbol{RT}(A)$ (Hyland, Johnstone, and Pitts [1980]); we will refer loosely to the categories $\boldsymbol{PER}(A)$, $\boldsymbol{RT}(A)$ and their close relatives as *realizability models*. Here we shall concentrate on their connections with programming languages and computability.

One way of thinking about the above definitions (advocated by Mitchell and frequently stressed by the present author) is to regard a PCA A as a kind of abstract model of "machine level" computation, and to regard $\boldsymbol{PER}(A)$ as a category of "datatypes", for a high level programming language implemented on this machine. In the case of K_1, for instance, this accords closely with what happens inside a computer: a high-level datatype consists of values which must ultimately be represented on the machine somehow by bit sequences (or let us say by natural numbers). Since two machine representations of some value might be indistinguishable from the point of view of the high-level language, we can think of the datatype as corresponding to a partial equivalence relation on $\mathbb{N}$. Abstracting from this situation, we can imagine any PCA A as providing a kind of primitive model of computation, and on this view all morphisms of $\boldsymbol{PER}(A)$ are "computable" in the sense that they are realized by an element of A.

5.1. Type structures in realizability models. In any category of the categories $\boldsymbol{PER}(A)$, there is (up to isomorphism) a canonical "datatype of natural numbers" N, arising for instance from Curry's encoding of the natural numbers in the language of combinatory logic. (We will write N for the object of $\boldsymbol{PER}(A)$ representing the natural numbers, to distinguish it from the ordinary set $\mathbb{N}$ of natural numbers.) In addition, for most of the particular PCAs of interest, there is an object in $\boldsymbol{PER}(A)$ that stands out as being the obvious choice for $N_\perp$. We can therefore obtain type structures over $\mathbb{N}$ and $\mathbb{N}_\perp$ by repeated exponentiation in $\boldsymbol{PER}(A)$. From our point of view, these can be seen as type structures of "computable functionals" naturally arising from the notion of computability embodied by A. In the cases where A is a genuinely "effective" PCA, this will yield type structures of effectively computable functionals in some sense. We therefore have a rich supply of interesting constructions of type structures to consider.

In fact, many of the characterizations of type structures that we have already considered are easily seen to be of precisely this kind. As regards total type structures over $\mathbb{N}$, for instance, Kreisel's definition of HEO (Definition 3.16) is nothing other than the definition of the type structure over N in $\boldsymbol{PER}(K_1)$. Kleene's definition of C via associates (Definition 3.13) essentially arises in the same way from a PCA known as *Kleene's second model* K_2. This PCA was introduced in Kleene and Vesley [1965] — here the underlying set is $\mathbb{N}^{\mathbb{N}}$, and application is given by a minor modification of the operation $(- \mid -)$ mentioned in Section 3.3.1. (In fact, an associate for an element $x \in \mathsf{C}$ is essentially nothing other than a realizer for x in K_2.) The definition of HRC (Definition 3.18) arises similarly from the recursive submodel K_{2rec}. Scott's characterization of C via algebraic lattices is very close to the definition of the type structure over N in the Scott *graph model* $\mathcal{P}\omega$.

Other constructions can naturally be viewed as type structures in other kinds of realizability models. For instance, Bezem's construction of HEO as the extensional collapse of HRO (Theorem 3.20) corresponds to the type structure over N in the category $\boldsymbol{MPER}(A)$ of *modified PERs* on A, a natural subcategory of the *modified realizability topos* on A (see Oosten [1997]). In addition, several possible definitions of RC correspond to type structures in *relative realizability* models of the kind considered in Awodey, Birkedal, and Scott [2000].

The general programme of trying to identify the type structures over $\mathbb{N}$ arising from various PCAs was explicitly articulated in Beeson [1985, Chapter VI], where the cases of $\mathcal{P}\omega$ and $\mathcal{P}\omega_{re}$ (giving rise to C and HRC respectively) were considered. Similar results for other graph models were obtained by Bethke (Bethke [1988]). As one might expect, it would seem that all natural PCAs based solely on a notion of continuous function application give rise to the type structure C, and their recursive analogues give rise to HRC.

One can also view many definitions of type structures over $\mathbb{N}_\perp$ as arising from realizability models, but for this we need some additional ideas. A suitable way of talking about "computable partial functions" in categories such as $\boldsymbol{PER}(A)$ was provided by Rosolini's theory of *dominances* (Rosolini [1986]), inspired by ideas of Mulry. A dominance is a small piece of extra structure on a category which determines an abstract notion of "semidecidable predicate". Under certain conditions, a dominance gives rise to a *lifting* operation $X \mapsto X_\perp$ on objects; one may then identify computable partial functions $X \to Y$ with morphisms $X \to Y_\perp$. For instance, the natural choice of dominance on $\boldsymbol{PER}(K_1)$ gives rise to an object $N_\perp$ which may be defined (as a PER) by

$$N_\perp(m,n) \iff \phi_m(0) \simeq \phi_n(0).$$

The morphisms $N \to N_\perp$ then correspond precisely to the partial recursive functions $\mathbb{N} \rightharpoonup \mathbb{N}$, as we might have hoped.

In Longley [1995], Longley developed explicitly the idea that different PCAs embody different notions of computability, and considered the problem of identifying the type structures over $N_\perp$ in particular models. The following result appeared in Longley [1995, Chapter 7]:

THEOREM 5.3. $\mathsf{T}(\boldsymbol{PER}(K_1), N_\perp) \cong \mathsf{P}^{\mathit{eff}}$.

This is very close in content as Ershov's generalized Myhill-Shepherdson theorem (Theorem 4.12); however, the formulation in terms of $\boldsymbol{PER}(K_1)$ is simpler insofar as here it is immediate that the required exponentials exist.

Longley drew particular attention to the coincidence between the type structure in $\boldsymbol{PER}(K_1)$ and the extensional term model for PCF^{++} (Section 4.2.5). Both of these constructions can be seen as very "pure" attempts at defining a class of hereditarily computable partial functionals (they do not require auxiliary concepts such as continuity), but they are totally different in spirit. In $\boldsymbol{PER}(K_1)$ we think of computations as taking place at the level of recursive indices, and an index realizes a functional whenever its action on indices for objects of lower type just "happens" to be extensional. In PCF^{++}, by contrast, computation takes place at a higher "symbolic" level, where extensionality is enforced by the design of the programming language, and the ways in which a function may interact with its argument seem *prima facie* to be much more restricted. That these two definitions yield the same structure at all finite types still appears to the present author to be a wonderful and surprising fact.

It was also shown in Longley [1995] that the type structure over (an obvious object) $N_\perp$ in $\boldsymbol{PER}(\mathcal{P}\omega)$ [resp. in $\boldsymbol{PER}(\mathcal{P}\omega_{re})$] coincided with P [resp. $\mathsf{P}^{\mathit{eff}}$].

This is not too surprising in view of the close connections between Scott domains and algebraic lattices.

Another interesting family of PCAs are the term models for untyped λ-calculi. For instance, let Λ^0/T be the set of pure closed untyped λ-terms modulo some reasonable theory T, regarded as a combinatory algebra. The following was conjectured implicitly (for a particular T) by Phoa (Phoa [1991]), and more explicitly (for a large class of theories T) by Longley (Longley [1995, Section 7.4]), who obtained some partial results and outlined a possible proof strategy:

CONJECTURE 5.4 (Longley-Phoa). *In the type structure over* (*an obvious choice of*) $N_\bot$ *in* $\boldsymbol{PER}(\Lambda^0/T)$, *every element is* PCF-*definable. In other words,*

$$\mathsf{T}(\boldsymbol{PER}(\Lambda^0/T), N_\bot) \cong \mathsf{Q}^{\mathit{eff}}.$$

There is fairly good evidence for this conjecture, but it has resisted extensive attempts at proof by the author and others. Moreover, even if proved, it is doubtful whether this result would give any useful information about the type structure $\mathsf{Q}^{\mathit{eff}}$, since the relevant objects in $\boldsymbol{PER}(\Lambda^0/T)$ appear to be quite intractable. The main interest in the conjecture is perhaps conceptual: it would provide an example of a highly non-trivial equivalence between two characterizations of $\mathsf{Q}^{\mathit{eff}}$, which in turn would provide evidence for the mathematical naturalness of this type structure. An interesting (and perhaps more useful) variant of the Longley-Phoa conjecture, involving a λ-calculus with certain constants, has been established by Streicher *et al* (Marz, Rohr, and Streicher [1999], Rohr [2002]), though this is a much easier result.

A more semantic example of a PCA which does give rise to $\mathsf{Q}^{\mathit{eff}}$ was constructed by Abramsky, based on the ideas of games and well-bracketed strategies. The proof that it does so is non-trivial and to some extent furnishes the same kind of evidence for the status of Q and $\mathsf{Q}^{\mathit{eff}}$ as would be provided by the Longley-Phoa conjecture; see Abramsky and Longley [2000].

Meanwhile, van Oosten and Longley constructed the PCA $\mathcal{B}$ described in Section 4.4; the type structure defined there is exactly the type structure over the obvious object $N_\bot$ in $\boldsymbol{PER}(\mathcal{B})$. It was this characterization that led to the systematic study of the sequentially realizable functionals in Longley [2002a].

Various attempts have been made to generalize the construction of realizability models from untyped to typed structures. One way to do this was proposed by the present author in Longley [1999b]; we will make considerable use of this perspective in Parts II and III as a means of unifying much of the material described in this survey. This generalization also leads to many new ways of constructing type structures, though in most cases they turn out to be isomorphic to one of the type structures already known.

In conclusion, it appears that in almost all known "natural" examples of realizability models, the type structure over N is isomorphic to one of C, RC

and HRC; while the type structure over (a natural choice of) $N_\perp$ is isomorphic to one of P, Q, R or their recursive analogues. This is consistent with the overall impression gained from the material in Sections 3 and 4 — namely, that a wide range of approaches to defining plausible notions of higher type computable functional actually leads to a relatively small and manageable handful of type structures. It would be interesting to know whether the Kleene computable functionals (*e.g.*, $\mathsf{KC}(A)$ for some A, or the type structure K mentioned in Section 4.3.1) can be obtained via a realizability construction in a natural way.

5.2. Domains in realizability models. Much of the work mentioned in the preceding section took place within the context of a general programme known as *synthetic domain theory*, which sought to identify good categories of "computational domains" (*e.g.*, for denotational semantics) within realizability models and similar categories. We include a short account of this programme here in order to give an impression of the background to the above results.

Synthetic domain theory was initiated by Scott in the early 1980s. The idea of this enterprise was to look for models of constructive set theory containing objects which, *by themselves* and without any additional structure, could serve as domains for denotational semantics. Domains would then simply be "sets" in some constructive universe, and the hope was that this might lead to a simpler, cleaner version of domain theory than the classical one.

Later work by Rosolini (Rosolini [1986]), Hyland (Hyland [1990]) and many other researchers sought to give axiomatic versions of synthetic domain theory: that is, a set of conditions on a model of constructive set theory (usually a topos) which suffice to ensure that it contains a good category of computational domains. Here realizability models provided the main source of motivating examples. Longley and Simpson (Longley and Simpson [1997]) developed a version of synthetic domain theory that applied uniformly to a wide range of realizability models, showing that very many PCAs gave rise to models of PCF at least. We refer to Rosolini's contribution in Fiore, Jung, Moggi, O'Hearn, Riecke, Rosolini, and Stark [1996] for a brief survey of synthetic domain theory and further references.

Related to this enterprise was the observation that many known categories of computational domains could be seen as full subcategories of particular realizability models. For instance, there is an obvious embedding of ***EN*** in $\boldsymbol{PER}(K_1)$ which preserves the exponentials that exist in ***EN***. Another example was provided by the work of McCarty (McCarty [1984]), who showed that the category of effectively given information systems (essentially equivalent to effective Scott domains) embeds fully in a model of intuitionistic ZF set theory based on Kleene realizability; this was recast in terms of $\boldsymbol{PER}(K_1)$ in Longley [1995].

Rather more easily, the category of Scott domains and its effective analogue can respectively be embedded in $\boldsymbol{PER}(\mathcal{P}\omega), \boldsymbol{PER}(\mathcal{P}\omega_{re})$. These categories play a central role in the work of Scott and his colleagues (Bauer, Birkedal, and Scott [2001]), who exploit the observation that PERs on $\mathcal{P}\omega$ are equivalent to countably based T_0 spaces equipped with an equivalence relation (such objects are termed *equilogical spaces*). The work of van Oosten and Longley on sequential realizability (Section 4.4) has shown that certain categories of sequential algorithms and hypercoherences arise as subcategories of $\boldsymbol{PER}(\mathcal{B})$. Finally, Bauer has recently shown (Bauer [2001]) that much of the work of Weihrauch *et al* on representations of spaces via type two effectivity (Section 3.3.5) can be naturally understood in terms of the categories $\boldsymbol{PER}(K_2)$, $\boldsymbol{PER}(K_{2rec})$. All these results suggest that realizability models can provide an attractive setting for describing and relating many other kinds of models.

§6. Non-functional notions of computability.

Thus far we have concentrated almost entirely on extensional notions of computability — that is, on notions of computable *functional*. One can also ask whether there are reasonable non-extensional notions of "computable operation" at higher types. Such notions have received relatively little attention by comparison with the extensional notions — perhaps because the very idea of an "intensional operation" seems rather hazy, and it is unclear *a priori* whether it is amenable to a precise mathematical formulation. We here briefly survey some known ideas that relate to this problem.

We have seen how notions of computable functional may be naturally embodied by extensional type structures (or substructures thereof). As a first attempt, therefore, we might propose that more general notions of computable operation could be identified simply with type structures without the extensionality requirement. A typical example would be the structure HRO of Definition 3.17. Many other examples arise from (non-well-pointed) cartesian closed categories: given any object X corresponding to N or $N_\perp$, interpret the simple types by repeated exponentiation and then apply the global elements functor $\mathrm{Hom}(1, -)$. This view seems somewhat unsatisfactory in that it is too concrete: for instance, different Gödel-numbering schemes can give rise to non-isomorphic variants of HRO, whereas we would presumably wish to consider all these variants as embodying essentially the same "notion of computability". This problem may be addressed by adopting the more refined point of view outlined in Section 6.3 below; in the meantime, however, we may at least collect examples of non-extensional type structures which might embody plausible notions of computable operation.

6.1. Structures over $\mathbb{N}$.

Non-extensional type structures over $\mathbb{N}$ were systematically studied by Troelstra (Troelstra [1973]), who exploited them for metamathematical purposes: any such type structure containing suitable basic operations (essentially those of System T) can serve as a model for higher order arithmetic without the extensionality axiom. Two of the type structures considered by Troelstra are of particular interest from our point of view: the structure HRO of hereditarily recursive operations, and a type structure ICF of *intensional continuous functionals*. Both structures were first introduced by Kreisel (Kreisel [1958], [1962]).

Many of the relevant facts about HRO have already been described in Section 3.4. Let us call an element $F \in \mathsf{HRO}_2$ an *extensional operation* if $F \cdot f = F \cdot f'$ whenever $f \cdot n = f' \cdot n$ for all n. A trivial example of a non-extensional operation is the element $G \in \mathsf{HRO}_2$ which given an element $x \in \mathsf{HRO}_1$ simply returns $x \in \mathsf{HRO}_0$. As a less trivial example of a non-extensional phenomenon at higher types, there is a *local modulus of continuity* operation $\Psi \in \mathsf{HRO}_{\overline{2}\to\overline{1}\to\overline{0}}$ with the following property: if $F \in \mathsf{HRO}_2$ is an extensional operation and $g \cdot n = g' \cdot n$ for all $n < \Psi \cdot F \cdot g$, then $F \cdot g = F \cdot g'$. (The existence of such a recursive operation is implicit in the original proof of the Kreisel-Lacombe-Shoenfield theorem; see Kreisel, Lacombe, and Shoenfield [1959].) By contrast, it is easily shown that no *extensional* local modulus of continuity operation is computable: that is, there is no element $\Psi \in \mathsf{HEO}$ with the above property. Observations such as these can be used to obtain a variety of consistency and independence results for theories of higher order arithmetic, and also to give us a feel for what operations are and are not computable in various settings.

The type structure ICF is the intensional counterpart of C (defined as in Definition 3.13) in the way that HRO is the intensional counterpart of HEO. That is, it is essentially obtained from Kleene's second model K_2 (see Section 5.1) in the same way in which HRO is obtained from K_1. For pure types, ICF may be defined as follows:

- $\mathsf{ICF}_0 = \mathbb{N}$, $\mathsf{ICF}_1 = \mathbb{N}^{\mathbb{N}}$;
- $\mathsf{ICF}_{n+1} = \{f \in \mathbb{N}^{\mathbb{N}} \mid \forall g \in \mathsf{ICF}_n.\ (f \mid g)\downarrow\}$

where $(f \mid g)$ is as defined in Section 3.3.1. As with the extensional type structures, one may also consider the recursive analogue of ICF, or its recursive substructure. It can be shown that ICF contains a local modulus of continuity operation as above, while C does not, and similarly for the recursive variants (see Troelstra [1973, §2.6]).

Troelstra also considered other non-extensional type structures, such as certain term models for System T, though these seem less appealing as candidates for notions of computability.

6.2. Structures over $\mathbb{N}_\perp$. Other non-extensional notions of computability have more recently been considered in computer science, where they arise naturally in connection with typed programming languages containing non-functional features such as *exceptions* or *state*. The term models for such programming languages frequently give rise to non-well-pointed cartesian closed categories, and hence to non-extensional type structures. One can regard such term models as defining notions of computability by themselves, though as in the extensional case, much of the interest lies in trying to provide other, more semantic characterizations of these type structures. Since the number of programming languages that have been considered in the computer science literature is very large, and for most of them little of interest is known from the point of view of computability, we will confine our attention here to languages for which some alternative characterization of the implicit notion of computability has been obtained.

One intensional notion of computability that has emerged as having good credentials is embodied by the language PCF+`catch` studied in Cartwright and Felleisen [1992], Cartwright, Curien, and Felleisen [1994], as well as by PCF with (first order) `callcc` (Kanneganti, Cartwright, and Felleisen [1993]) and by μ PCF (Ong and Stewart [1997]). All these languages are equivalent in the sense that their fully abstract term models (consisting of closed terms modulo observational equivalence) are isomorphic at the finite types. Here we will give a definition of PCF +`catch` as a representative of these languages:

Definition 6.1 (PCF +`catch`).

(i) *Define the syntax of* PCF +`catch` *by augmenting the definition of* PCF (*Definition* 4.16) *with a constant* $\texttt{catch}_r : (\overline{0}^r \to \overline{0}) \to \overline{0}$ *for each* $r > 0$.

(ii) *Define a* context $E[-]$ *to be a* PCF +`catch` *term with a single occurrence of a hole '$-$' (in an obvious sense). The* evaluation contexts *are generated inductively as follows: $(-)$ is an evaluation context; and if $E[-]$ is an evaluation context then so are*

$$\texttt{succ}\,E[-],\ \ \texttt{pred}\,E[-],\ \ \texttt{if}\,E[-],\ \ E[-]V,\ \ \texttt{catch}_r(\lambda x_1 \cdots x_r.E[-]).$$

(*Intuitively, if $E[-]$ is an evaluation context, then in order to evaluate a term $E[U]$ the subterm U needs to be evaluated "next".*)

(iii) *Let $\rightsquigarrow$ be the smallest transitive relation on terms of* PCF +`catch` *satisfying the clauses for PCF (see Definition* 4.18) *together with the following two clauses:*

- *If $Ux_0 \dots x_{r-1} \rightsquigarrow E[x_i]$ where the $x_j : \overline{0}$ are fresh variables and $E[-]$ is an evaluation context (intuitively, if we cannot proceed further with the reduction without knowing x_i), then $\texttt{catch}_r\ U \rightsquigarrow \widehat{i}$.*
- *If $Ux_0 \dots x_{r-1} \rightsquigarrow \widehat{k}$, where the $x_j : \overline{0}$ are fresh variables (intuitively, if we can complete the computation without knowing any of the x_i), then $\texttt{catch}_r\ U \rightsquigarrow \widehat{r+k}$.*

Informally, $\texttt{catch}_r\ U$ evaluates $U\,x_0 \dots x_{r-1}$ and watches to see if U ever has to look at one of the x_i; if so, the computation is aborted and the index for the argument looked at is returned. It is easy to see how the functional F of Section 4.4 can be encoded in PCF+`catch`. Unlike F, however, the catch operators give rise to non-functional behaviour: for instance, we have

$$\texttt{catch}_2(\lambda xy.\, x + y) \rightsquigarrow 0, \quad \texttt{catch}_2(\lambda xy.\, y + x) \rightsquigarrow 1.$$

The fact that several proposed languages turn out to have the same expressivity as PCF+`catch` is already encouraging, but more significant is the following semantic characterization due to Cartwright, Curien and Felleisen (Cartwright, Curien, and Felleisen [1994]):

THEOREM 6.2. *The fully abstract term model for* PCF +`catch` *is isomorphic to the type structure over* $\mathbb{N}_\perp$ *arising from the category of effective sequential algorithms* (*that is, the effective analogue of the Berry-Curien model*; *see Berry and Curien* [1982]).

The original definition of the sequential algorithms model is rather heavy and we will not give it here. However, Longley showed that the relevant part of this model can be easily reconstructed from the van Oosten algebra $\mathcal{B}$ (Longley [2002a, §4]); this gives a simpler alternative description of the type structure of Theorem 6.2.

Since then, the ideas of game semantics as developed by Abramsky *et al* have been successful in providing a semantic account of the expressivity of a number of programming languages. In Abramsky and McCusker [1999], for instance, it is shown that by imposing or not imposing the well-bracketing and innocence conditions on strategies (see Section 4.3.3) one can obtain a square of four categories of games corresponding to different computational paradigms, of which only one corresponds to a functional notion of computation (namely the model with both well-bracketing and innocence constraints, which corresponds to PCF). The model with innocence but not well-bracketing (studied in detail in Laird [1998], [1997]) turns out to correspond to PCF+`catch`: more precisely, the fully abstract term model for PCF+`catch` is a quotient of the type structure arising from this model. The models without innocence correspond to notions of computation involving memory or state; a full abstraction result for Idealized Algol (with respect to the non-innocent, well-bracketed game model) is obtained in Abramsky, Honda, and McCusker [1998].

In general, it would appear that categories of games and their correlations with programming languages provide a fruitful source of candidates for mathematically natural notions of non-extensional computability, and we expect more progress in this vein in the near future.

6.3. A realizability perspective. We have concentrated here on non-extensional type structures as a way to capture notions of computable operation. A somewhat more subtle perspective, making use of ideas from realizability, was outlined in Longley [1999a], [1999b]. The idea here is that given a realizability interpretation for a logic (say predicate logic for the simple types over $\mathbb{N}$ or $\mathbb{N}_\perp$) in which the realizers are drawn from some computational universe A (such as a PCA or a type structure of some kind), the set of realizable sentences gives information about what kinds of operation are computable in A. For example, whether the sentence

$$\forall F : \overline{2}.\ \forall g : \overline{1}.\ \exists n : \overline{0}.\ \forall g' : \overline{1}.\ (\forall m < n.\ g(m) = g'(m)) \Rightarrow F(g) = F(g')$$

is realizable in A corresponds to whether local moduli of continuity are computable in A (see Section 6.1). We might propose, therefore, to *identify* notions of higher type computability with notions of realizability for such a logic. In some sense, this allows us to say what intensional operations are computable without having to commit ourselves to any concrete definition of "computable operation". This perspective will be further advocated and developed in detail in Part III.

This point of view is very general and leads to a large class of potential notions of computability: for instance, any untyped PCA implicitly embodies a notion of computable operation at higher types. Fortunately, however, the theory also allows us to say when two structures are equivalent from the point of view of computability, and this significantly cuts down the number of distinct notions to be considered. Even so, there are many notions of computability here competing for our attention, and much territory remains to be explored. It seems unclear as yet whether it is reasonable to hope ultimately for a small collection of genuinely natural notions, such as we have in the extensional setting, or whether there is in practice an unlimited range of plausible notions.

§7. Conclusion and prospectus. In this paper we have tried to trace the various lines of research to date that are relevant to the study of computability at higher types. Although these strands of research have been rather widely scattered across different areas of mathematical logic and computer science, it is our contention that when viewed together, the outline of a coherent subject area may be discerned.

The central conceptual question we are concerned with is "What are the good notions of computability at higher types, if there are any?" Our approach in this paper has been rather empirical: to collect a variety of different attempts at defining such a notion, and see what natural notions emerge. Our main concern has been to chart the history of the ideas that bear on the problem. The development of these ideas has (naturally enough) been rather haphazard, and we have made little attempt here to organize the material beyond what has been necessary to tell a coherent story.

Having collected and reviewed all this material, we are in a position to undertake a more systematic treatment. We will attempt this in the remaining two papers of the series, where we will try to show that the situation is less chaotic than it may at times have seemed during the course of this survey.

In Part II we will survey the material on notions of computable *functional* (that is, extensional notions of computability) within a uniform framework, presenting the important notions of computability, their various characterizations, their intrinsic properties, the relationships between different notions, and some discussion of their conceptual status. We will also include some results relating to the impossibility of a "Church's thesis" for higher type functionals.

We will argue that almost everything of interest that is known in the subject can be understood in terms of six basic notions of computability, which are represented in the world of "effective" type structures by RC, HRC, $\mathsf{P}^{\mathit{eff}}$, $\mathsf{Q}^{\mathit{eff}}$, $\mathsf{R}^{\mathit{eff}}$ and $\mathsf{K}^{\mathit{eff}}$ (the last of these being the type structure mentioned in Section 4.3.1, which will suffice to account for Kleene computability over all structures of interest). It seems to us probable that the picture here is by now reasonably complete: the territory has been fairly thoroughly explored, and all reasonable definitions of a natural class of effectively computable functionals seem to lead to one of the above notions.

Regarding more general notions of computable operation, the overall picture seems much less complete at present, but we are at least able to bring together a range of possible notions within a unified framework. In Part III we will develop in more detail the realizability perspective mentioned in Section 6.3, and will collect and organize much of what is known within this framework. We will see that a wide variety of results from recursion theory and computer science can be conveniently encapsulated in this setting. The framework used in Part III will in one sense subsume that of Part II, but the flavour of the two treatments will be rather different and we believe both to be valuable.

7.1. Acknowledgements. I am very grateful to the organizers of the Logic Colloquium for giving me the opportunity to lecture on this material, for providing me with the stimulus to write this article, for willingly tolerating its inordinate length, and for waiting so long for me to finish it. Many people have helped me to fill gaps in my knowledge of the subject, including Samson Abramsky, Ulrich Berger, Yuri Ershov, Solomon Feferman, Martin Hyland, Dag Normann, Gordon Plotkin, Helmut Schwichtenberg, Dana Scott, Alex Simpson and Stan Wainer. Thomas Streicher provided detailed comments on an earlier version of the paper, as did the anonymous referee who made many helpful suggestions for improving its clarity. Helpful feedback on later versions was provided by Samson Abramsky, Yiannis Moschovakis, Dag Normann, Jim Royer and Alex Simpson. I am also grateful to Florrie Kemp for her friendship.

During the writing of this paper I was supported by EPSRC Research Grants GR/L89532 "Notions of computability for general datatypes" and GR/N64571 "A proof system for correct program development".

FIGURE 3. Extensional type structures over $\mathbb{N}$.

$$\begin{tikzcd}
\mathsf{S} & \mathsf{C} \arrow[l, dashed] & \\
& \mathsf{RC} \arrow[u] & \mathsf{HRC} \cong \mathsf{HEO} \\
\mathsf{KC(S)} \arrow[uu] \arrow[r, dashed] & \mathsf{KC(C)} \arrow[u] & \mathsf{KC(HRC)} \arrow[u] \\
& \mathsf{T} \arrow[ul] \arrow[u] \arrow[ur] & \\
& \widehat{\mathsf{T}} \simeq \mathrm{S1\text{--}S8} \arrow[u] &
\end{tikzcd}$$

Appendix A. Summary of type structures. For reference, we include here two diagrams summarizing the main type structures we have considered in this article, and the principal relationships between them. For the sake of clarity we show only the extensional type structures. We give separate diagrams for the type structures over $\mathbb{N}$ (Figure 3) and the type structures over $\mathbb{N}_\perp$ (Figure 4). For the purpose of the diagrams, we identify a language such as System T or PCF with its extensional term model (all the languages concerned have a unique non-trivial such model). Definitions of all these structures and languages in the main text may be located using the index of symbols.

The arrows in the diagrams correspond to morphisms of the following kind:

DEFINITION A.1. *Suppose A, B are type structures over X. A* morphism *$R\colon A \to B$ is a family of total relations R_σ from A_σ to B_σ such that $R_0 = \mathrm{id}_X$ and*

$$R_{\sigma\to\tau}(f, f') \wedge R_\sigma(x, x') \Longrightarrow R_\tau(f \cdot x, f' \cdot x')$$

These morphisms will be studied at length in Part II. For extensional type structures they may be thought of loosely as "embeddings", though not all of them are straightforward substructure inclusions.

The broken arrows represent morphisms that exist but do not seem to be particularly significant. A few other morphisms which we consider unimportant have been omitted.

FIGURE 4. Extensional type structures over $\mathbb{N}_\perp$.

$$\begin{tikzcd}
\mathsf{M} & & \\
\mathsf{P} \arrow[u] & & \mathsf{R} \\
\mathrm{PCF}^{++} \simeq \mathsf{P}^{\mathit{eff}} \arrow[u] & \mathsf{Q} \arrow[ul] \arrow[ur] & \mathsf{R}^{\mathit{eff}} \simeq \mathrm{PCF} + \mathtt{H} \arrow[u] \\
& \mathsf{Q}^{\mathit{eff}} \simeq \mathrm{PCF} \arrow[ul] \arrow[u] \arrow[ur] & \\
& \mathsf{K}^{\mathit{eff}} \arrow[u] &
\end{tikzcd}$$

Appendix B. Remarks on bibliography. In the following list of references, we have attempted to provide reasonably comprehensive bibliography for the field of higher type computability as delineated by this article. One of our aims in this survey has been to provide an accessible guide to the literature of the subject, and since almost all the works listed below are cited somewhere in the text, it is possible to view the entire article as an extended commentary on the bibliography.

We have tried to include as many relevant published works as possible, along with any Ph.D. theses and other unpublished works that made important contributions to the subject. (The author would appreciate being informed of any significant omissions.) Unpublished technical reports whose contributions appeared soon afterwards in the published literature are usually not listed.

We have not attempted complete coverage of related areas in which ideas from higher type computability are applied (see Section 1.1), including only a selection of the more important works on these topics. For instance, there is a large literature on real number computability and computable analysis, of which only a few works have been included. We have given priority here to very early papers, and to works concerned explicitly with higher types over the reals.

We rather regret that we have not had the time for a fuller coverage of the literature on complexity at higher types. Fortunately, a thorough survey of this area has recently appeared in Irwin, Kapron, and Royer [2001a], [2001b].

Many helpful survey articles on closely related topics, and further references, can be found in Griffor [1999]. For an extensive bibliography on realizability, see Birkedal [1999].

REFERENCES

O. Aberth [1980], ***Computable analysis***, McGraw-Hill, New York.

S. Abramsky, K. Honda, and G. McCusker [1998], *A fully abstract game semantics for general references*, ***Proceedings of 13th Annual Symposium on Logic in Computer Science***, IEEE, pp. 334–344.

S. Abramsky, R. Jagadeesan, and P. Malacaria [2000], *Full abstraction for PCF*, ***Information and Computation***, vol. 163, pp. 409–470.

S. Abramsky and J. R. Longley [2000], *Some combinatory algebras for sequential computation*, in preparation.

S. Abramsky and G. McCusker [1999], *Game semantics*, ***Computational Logic: Proceedings of the 1997 Marktoberdorf summer school*** (H. Schwichtenberg and U. Berger, editors), Springer-Verlag, pp. 1–56.

P. Aczel [1977], *An introduction to inductive definitions*, ***Handbook of mathematical logic*** (J. Barwise, editor), North-Holland, pp. 739–782.

P. Aczel and P. G. Hinman [1974], *Recursion in the superjump*, ***Generalized Recursion Theory*** (J. E. Fenstad and P. G. Hinman, editors), North-Holland, pp. 3–41.

J. Avigad and S. Feferman [1998], *Gödel's functional ("Dialectica") interpretation*, ***Handbook of proof theory*** (S. R. Buss, editor), North-Holland, pp. 337–405.

S. Awodey, L. Birkedal, and D. S. Scott [2000], *Local realizability toposes and a modal logic for computability*, to appear in ***Mathematical Structures in Computer Science***.

S. Banach and S. Mazur [1937], *Sur les fonctions calculables*, ***Annales de la Société Polonaise de Mathématique***, vol. 16, p. 223.

H. P. Barendregt [1984], ***The lambda calculus: Its syntax and semantics***, revised ed., Studies in Logic and the Foundations of Mathematics, vol. 103, North-Holland.

H. P. Barendregt [1992], *Lambda calculi with types*, ***Handbook of logic in computer science*** (S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors), vol. 2, Oxford University Press, pp. 117–309.

A. Bauer [2000], *The realizability approach to computable analysis and topology*, ***Ph.D. thesis***, School of Computer Science, Carnegie Mellon University.

A. Bauer [2001], *A relationship between equilogical spaces and Type Two Effectivity*, ***Electronic Notes in Theoretical Computer Science*** (S. Brooks and M. Mislove, editors), vol. 45, Elsevier Science Publishers.

A. Bauer and L. Birkedal [2000], *Continuous functionals of dependent types and equilogical spaces*, ***Proceedings of Computer Science Logic 2000*** (P. G. Clote and H. Schwichtenberg, editors), Lecture Notes in Computer Science, vol. 1862, Springer-Verlag, pp. 202–216.

A. Bauer, L. Birkedal, and D. S. Scott [2001], *Equilogical spaces*, to appear in ***Theoretical Computer Science***.

A. Bauer, M. H. Escardó, and A. Simpson [2002], *Comparing functional paradigms for exact real-number computation*, to appear in ***Proceedings of International Conference on Automata, Languages and Programming***.

M. Beeson [1985], ***Foundations of constructive mathematics***, Springer-Verlag.

S. Bellantoni, K.-H. Niggl, and H. Schwichtenberg [2000], *Higher-type recursion, ramification and polynomial time*, ***Annals of Pure and Applied Logic***, vol. 104, pp. 17–30.

U. Berger [1990], *Totale Objekte und Mengen in der Bereichstheorie*, ***Ph.D. thesis***, Munich.

U. Berger [1993], *Total sets and objects in domain theory*, ***Annals of Pure and Applied Logic***, vol. 60, pp. 91–117.

U. Berger [1997], *Continuous functionals of dependent and transfinite types*, ***Technical report***, Habilitationsschrift, Ludwig-Maximilians-Universität München.

U. Berger [2000], *Minimisation vs. recursion on the partial continuous functionals*, draft paper.

U. Berger and H. Schwichtenberg [1991], *An inverse of the evaluation functional for typed λ-calculus*, ***Proceedings of 6th Annual Symposium on Logic in Computer Science***, IEEE, pp. 203–211.

J. A. Bergstra [1976], *Computability and continuity in finite types*, ***Ph.D. thesis***, University of Utrecht.

J. A. Bergstra [1978], *The continuous functionals and ^2E*, ***Generalized Recursion Theory II*** (J. E. Fenstad, R. O. Gandy, and G. E. Sacks, editors), North-Holland, pp. 39–53.

J. A. Bergstra and S. S. Wainer [1977], *The "real" ordinal of the 1-section of a continuous functional* (*abstract*), ***The Journal of Symbolic Logic***, vol. 42, p. 440.

G. Berry [1978], *Stable models of typed lambda-calculi*, ***Proceedings of 5th International Colloquium on Automata, Languages and Programming***, Lecture Notes in Computer Science, vol. 62, Springer-Verlag, pp. 72–89.

G. Berry and P.-L. Curien [1982], *Sequential algorithms on concrete data structures*, ***Theoretical Computer Science***, vol. 20, no. 3, pp. 265–321.

I. Bethke [1988], *Notes on partial combinatory algebras*, ***Ph.D. thesis***, University of Amsterdam.

M. Bezem [1985a], *Isomorphisms between HEO and HRO$^{\mathrm{E}}$, ECF and ICF$^{\mathrm{E}}$*, ***The Journal of Symbolic Logic***, vol. 50, pp. 359–371.

M. Bezem [1985b], *Strongly majorizable functionals of finite type: a model for bar recursion containing discontinuous functionals*, ***The Journal of Symbolic Logic***, vol. 50, pp. 652–660.

M. Bezem [1988], *Equivalence of bar recursors in the theory of functionals of finite type*, ***Archive of Mathematical Logic***, vol. 27, pp. 149–160.

M. Bezem [1989], *Compact and majorizable functionals of finite type*, ***The Journal of Symbolic Logic***, vol. 54, pp. 271–280.

L. Birkedal [1999], *Bibliography on realizability*, ***Proceedings of Workshop on Realizability, Trento*** (L. Birkedal, J. van Oosten, G. Rosolini, and D. S. Scott, editors), published as Electronic Notes in Theoretical Computer Science 23 No. 1, Elsevier. Available via `http://www.elsevier.nl/locate/entcs/volume23.html`.

B. Bloom and J. G. Riecke [1989], *LCF should be lifted*, ***Proceedings of Conference on Algebraic Methodology and Software Technology***, Department of Computer Science, University of Iowa.

A. Bucciarelli [1993a], *Another approach to sequentiality: Kleene's unimonotone functions*, ***Proceedings of 9th Symposium on Mathematical Foundations of Programming Semantics, New Orleans***, Lecture Notes in Computer Science, vol. 802, Springer-Verlag, pp. 333–358.

A. Bucciarelli [1993b], *Sequential models of PCF: some contributions to the domain-theoretic approach to full abstraction*, ***Ph.D. thesis***, Dipartimento di Informatica, Università di Pisa.

A. Bucciarelli [1995], *Degrees of parallelism in the continuous type hierarchy*, ***Proceedings of 9th International Conference on Mathematical Foundations of Programming Semantics***.

A. Bucciarelli and T. Ehrhard [1991a], *Extensional embedding of a strongly stable model of PCF*, ***Proceedings of 18th International Conference on Automata, Languages and Programming, Madrid***, Lecture Notes in Computer Science, vol. 510, Springer-Verlag, pp. 35–46.

A. Bucciarelli and T. Ehrhard [1991b], *Sequentiality and strong stability*, ***Proceedings of 6th Annual Symposium on Logic in Computer Science***, IEEE, pp. 138–145.

A. Bucciarelli and T. Ehrhard [1993], *A theory of sequentiality*, ***Theoretical Computer Science***, vol. 113, pp. 273–291.

A. Bucciarelli and T. Ehrhard [1994], *Sequentiality in an extensional framework*, ***Information and Computation***, vol. 110, pp. 265–296.

R. Cartwright, P.-L. Curien, and M. Felleisen [1994], *Fully abstract semantics for observably sequential languages*, ***Information and Computation***, vol. 111, no. 2, pp. 297–401.

R. Cartwright and M. Felleisen [1992], *Observable sequentiality and full abstraction*, ***Proceedings of 19th Symposium on Principles of Programming Languages***, ACM Press, pp. 328–342.

A. Church [1936], *An unsolvable problem of elementary number theory*, ***American Journal of Mathematics***, vol. 58, pp. 345–363.

A. Church [1940], *A formulation of the simple theory of types*, ***The Journal of Symbolic Logic***, vol. 5, pp. 56–68.

D. A. Clarke [1964], ***Hierarchies of predicates of finite types***, Memoirs of the American Mathematical Society, vol. 51, American Mathematical Society.

L. Colson and T. Ehrhard [1994], *On strong stability and higher-order sequentiality*, ***Proceedings of 9th Annual Symposium on Logic in Computer Science***, IEEE, pp. 103–108.

S. Cook [1990], *Computability and complexity of higher type functions*, ***Proceedings of MSRI workshop on Logic from Computer Science*** (Y. Moschovakis, editor), Springer-Verlag, pp. 51–72.

S. B. Cooper [1999], *Clockwork or Turing U/universe? — Remarks on causal determinism and computability*, ***Models and Computability*** (S. B. Cooper and J. K. Truss, editors), Cambridge University Press.

P.-L. Curien [1993], ***Categorical combinators, sequential algorithms and functional programming***, second ed., Birkhäuser.

N. J. Cutland [1980], ***Computability***, Cambridge University Press.

M. Davis [1958], ***Computability and unsolvability***, McGraw-Hill.

M. Davis [1959], *Computable functionals of arbitrary finite type*, ***Constructivity in Mathematics: Proceedings of the colloquium held at Amsterdam, 1957*** (A. Heyting, editor), North-Holland, pp. 281–284.

J.-P. van Draanen [1995], *Models for simply typed lambda-calculi with fixed point combinators and enumerators*, ***Ph.D. thesis***, Catholic University of Nijmegen.

A. G. Dragalin [1968], *The computation of primitive recursive terms of finite type, and primitive recursive realization*, ***Zapiski Nauchnykh Seminarov Leningradskogo otdeleniia Matematicheskogo Instituta imeni V. A. Steklova***, vol. 8, pp. 32–45, translation in ***Seminars in Mathematics, V. A. Steklov Mathematical Institute, Leningrad, vol. 8 (1970), pp. 13–18***.

T. Ehrhard [1993], *Hypercoherences: a strongly stable model of linear logic*, ***Mathematical Structures in Computer Science***, vol. 3, pp. 365–385.

T. Ehrhard [1996], *Projecting sequential algorithms on strongly stable functions*, ***Annals of Pure and Applied Logic***, vol. 77, pp. 201–244.

T. Ehrhard [1999], *A relative PCF-definability result for strongly stable functions and some corollaries*, ***Information and Computation***, vol. 152, no. 1, pp. 111–137.

Yu.L. Ershov [1971a], *Computable numerations of morphisms*, ***Algebra i Logika***, vol. 10, no. 3, pp. 247–308.

Yu.L. Ershov [1971b], *La théorie des énumérations*, ***Actes du Congrès International des Mathématiciens, Nice 1970, Tome 1***, Gauthier-Villars, Paris, pp. 223–227.

Yu.L. Ershov [1972], *Computable functionals of finite type*, ***Algebra i Logika***, vol. 11, no. 4, pp. 203–277, English translation in ***Algebra and Logic***, vol. 11 (1972), 203–242, AMS.

Yu.L. Ershov [1973a], *Theorie der Numerierungen I*, ***Zeitschrift für mathematische Logik***, vol. 19, no. 4, pp. 289–388.

Yu.L. Ershov [1973b], *The theory of A-spaces*, ***Algebra i Logika***, vol. 12, no. 4, pp. 369–416, English translation in ***Algebra and Logic***, vol. 12 (1973), 209–232, AMS.

Yu.L. Ershov [1974a], *Maximal and everywhere defined functionals*, ***Algebra i Logika***, vol. 13, no. 4, pp. 210–255, English translation in ***Algebra and Logic***, vol. 13 (1974), 210–225, AMS.

Yu.L. Ershov [1974b], *On the model G of the theory BR*, ***Soviet Mathematics, Doklady***, vol. 15, no. 4, pp. 1158–1160.

Yu.L. Ershov [1975], *Theorie der Numerierungen II*, ***Zeitschrift für mathematische Logik***, vol. 21, no. 6, pp. 473–584.

Yu.L. Ershov [1976a], *Constructions 'by finite'*, ***Proceedings of 5th International Congress on Logic, Methodology and Philosophy of Science***, London, Ontario, pp. 3–9.

Yu.L. Ershov [1976b], *Hereditarily effective operations*, ***Algebra i Logika***, vol. 15, no. 6, pp. 642–654, English translation in ***Algebra and Logic***, AMS.

Yu.L. Ershov [1977a], *Model C of the partial continuous functionals*, ***Logic Colloquium 1976***, North-Holland, pp. 455–467.

Yu.L. Ershov [1977b], *Theorie der Numerierungen III*, ***Zeitschrift für mathematische Logik***, vol. 23, no. 4, pp. 289–371.

Yu.L. Ershov [1977c], ***The theory of enumerations***, Monographs in Mathematical Logic and Foundations of Mathematics, Nauka, Moscow.

Yu.L. Ershov [1996], ***Definability and computability***, Siberian School of Algebra and Logic, Plenum Publishing Corporation.

Yu.L. Ershov [1999], *Theory of numberings*, ***Handbook of computability theory*** (E. R. Griffor, editor), North-Holland, pp. 473–503.

M. H. Escardó [1996], *PCF extended with real numbers*, ***Theoretical Computer Science***, vol. 162, pp. 79–115.

S. Feferman [1975], *A language and axioms for explicit mathematics*, ***Algebra and logic*** (J. N. Crossley, editor), Springer-Verlag, pp. 87–139.

S. Feferman [1977a], *Inductive schemata and recursively continuous functionals*, ***Logic Colloquium 1976***, North-Holland, pp. 373–392.

S. Feferman [1977b], *Theories of finite type related to mathematical practice*, ***Handbook of mathematical logic*** (J. Barwise, editor), North-Holland, pp. 913–971.

S. Feferman [1996], *Computation on abstract datatypes. The extensional approach, with an application to streams*, ***Annals of Pure and Applied Logic***, vol. 81, pp. 75–113.

J. E. Fenstad [1978], *On the foundation of general recursion theory: Computations versus inductive definability*, ***Generalized Recursion Theory II*** (J. E. Fenstad, R. O. Gandy, and G. E. Sacks, editors), North-Holland, pp. 99–110.

J. E. Fenstad [1980], ***General recursion theory***, Perspectives in Mathematical Logic, Springer.

M. P. Fiore, A. Jung, Moggi, O'Hearn, Riecke, Rosolini, and Stark [1996], *Domains and denotational semantics: History, accomplishments and open problems*, ***Bulletin of the European Association for Theoretical Computer Science***, vol. 59, pp. 227–256.

M. C. Fitting [1981], ***Fundamentals of generalized recursion theory***, Studies in Logic and the Foundations of Mathematics, vol. 105, North-Holland.

R. V. Freivalds [1978], *Effective operations and functionals computable in the limit*, ***Zeitschrift für mathematische Logik und Grundlagen der Mathematik***, vol. 24, pp. 193–206.

R. M. Friedberg [1958a], *Four quantifier completeness: a Banach-Mazur functional not uniformly partial recursive*, ***Bulletin de l'Académie Polonaise des Sciences, Série des sciences mathématiques, astronomiques et physiques***, vol. 6, pp. 1–5.

R. M. Friedberg [1958b], *Un contre-exemple relatif aux fonctionnelles récursives*, ***Comptes rendus hebdomadaires des séances de l'Académie des Sciences (Paris)***, vol. 247, pp. 852–854.

H. M. Friedman [1971], *Algorithmic procedures, generalized Turing algorithms, and elementary recursion theory*, ***Logic Colloquium 1969*** (R. O. Gandy and C. E. M. Yates, editors), North-Holland, pp. 113–137.

R. O. Gandy [1962], *Effective operations and recursive functionals (abstract)*, ***The Journal of Symbolic Logic***, vol. 27, pp. 378–379.

R. O. Gandy [1967a], *Computable functionals of finite type I*, ***Sets, Models and Recursion Theory*** (J. N. Crossley, editor), North-Holland, part II never appeared, pp. 202–242.

R. O. GANDY [1967b], *General recursive functionals of finite type and hierarchies of functionals*, ***Annales de la Faculté des Sciences, Université de Clermont-Ferrand***, vol. 35, pp. 5–24.

R. O. GANDY [1980], *Proofs of strong normalization*, ***To H. B. Curry: Essays on combinatory Logic, lambda calculus and formalism*** (J. R. Hindley and J. P. Seldin, editors), Academic Press.

R. O. GANDY AND J. M. E. HYLAND [1977], *Computable and recursively countable functions of higher type*, ***Logic Colloquium 1976***, North-Holland, pp. 407–438.

P. GIANNINI AND G. LONGO [1984], *Effectively given domains and lambda-calculus models*, ***Information and Control***, vol. 62, pp. 36–63.

J.-Y. GIRARD [1972], *Interprétation functionelle et élimination des coupures de l'arithmétique d'ordre supérieur*, ***Ph.D. thesis***, Paris.

J.-Y. GIRARD [1987], ***Proof theory and logical complexity***, vol. I, Bibliopolis, volume II has not appeared.

J.-Y. GIRARD [1988], *Normal functors, power series and λ-calculus*, ***Annals of Pure and Applied Logic***, vol. 37, no. 2, pp. 129–177.

K. GÖDEL [1931], *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I*, ***Monatshefte für Mathematik und Physik***, vol. 38, pp. 173–198, English translation in J. van Heijenoort, ed., ***From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931***, Harvard University Press, 1967, pp. 596–616.

K. GÖDEL [1958], *Über eine bisher noch nicht Erweiterung des finiten Standpunktes*, ***Dialectica***, vol. 12, pp. 280–287, English translation in Gödel [1990].

K. GÖDEL [1972], *On an extension of finitary mathematics which has not yet been used*, First published in Gödel [1990].

K. GÖDEL [1990], ***Collected works***, vol. II, Oxford University Press, edited by S. Feferman *et al.*

E. R. GRIFFOR [1980], *E-recursively enumerable degrees*, ***Ph.D. thesis***, MIT, Cambridge, Massachusetts.

E. R. Griffor (editor) [1999], ***Handbook of computability theory***, Studies in Logic and the Foundations of Mathematics, vol. 140, North-Holland.

T. J. GRILLIOT [1967], *Recursive functions of finite higher types*, ***Ph.D. thesis***, Duke University.

T. J. GRILLIOT [1969a], *Hierarchies based on objects of finite type*, ***The Journal of Symbolic Logic***, vol. 34, pp. 177–182.

T. J. GRILLIOT [1969b], *Selection functions for recursive functionals*, ***Notre Dame Journal of Formal Logic***, vol. X, pp. 225–234.

T. J. GRILLIOT [1971], *On effectively discontinuous type-2 objects*, ***The Journal of Symbolic Logic***, vol. 36, pp. 245–248.

A. GRZEGORCZYK [1955a], *Computable functionals*, ***Fundamenta Mathematicae***, vol. 42, pp. 168–202.

A. GRZEGORCZYK [1955b], *On the definition of computable functionals*, ***Fundamenta Mathematicae***, vol. 42, pp. 232–239.

A. GRZEGORCZYK [1957], *On the definitions of computable real continuous functions*, ***Fundamenta Mathematicae***, vol. 44, pp. 61–71.

A. GRZEGORCZYK [1964], *Recursive objects in all finite types*, ***Fundamenta Mathematicae***, vol. 54, pp. 73–93.

L. A. HARRINGTON [1973], *Contributions to recursion theory in higher types*, ***Ph.D. thesis***, MIT, Cambridge, Massachusetts.

L. A. HARRINGTON [1974], *The superjump and the first recursively Mahlo ordinal*, ***Generalized Recursion Theory*** (J. E. Fenstad and P. G. Hinman, editors), North-Holland, pp. 43–52.

L. A. HARRINGTON AND A. S. KECHRIS [1975], *On characterizing Spector classes*, ***The Journal of Symbolic Logic***, vol. 40, pp. 19–24.

L. A. HARRINGTON AND D. MACQUEEN [1976], *Selection in abstract recursion theory*, ***The Journal of Symbolic Logic***, vol. 41, pp. 153–158.

J. P. Helm [1971], *On effectively computable operators*, ***Zeitschrift für mathematische Logik und Grundlagen der Mathematik***, vol. 17, pp. 231–244.

L. Henkin [1950], *Completeness in the theory of types*, ***The Journal of Symbolic Logic***, vol. 15, no. 2, pp. 81–91.

D. Hilbert [1925], *Über das Unendliche*, ***Mathematische Annalen***, vol. 95, pp. 161–190, English translation in J. van Heijenoort, ed., ***From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931***, Harvard University Press, 1967, pp. 367–392.

S. Hinata [1967], *Calculability of primitive recursive functionals of finite type*, ***Science Reports of the Tokyo Kyoiku Daigaku, A***, vol. 9, pp. 218–235.

S. Hinata and S. Tugué [1969], *A note on continuous functionals*, ***Annals of the Japan Association for Philosophy of Science***, vol. 3, pp. 138–145.

Y. Hinatani [1966], *Calculabilité des fonctionnels recursives primitives de type fini sur les nombres naturels*, ***Annals of the Japan Association for Philosophy of Science***, vol. 3, pp. 19–30.

P. G. Hinman [1966], *Ad astra per aspera: hierarchy schemata in recursive function theory*, ***Ph.D. thesis***, University of California, Berkeley.

P. G. Hinman [1969], *Hierarchies of effective descriptive set theory*, ***Transactions of the American Mathematical Society***, vol. 142, pp. 111–140.

P. G. Hinman [1973], *Degrees of continuous functionals*, ***The Journal of Symbolic Logic***, vol. 38, pp. 393–395.

P. G. Hinman [1978], ***Recursion-theoretic hierarchies***, Perspectives in Mathematical Logic, Springer-Verlag.

P. G. Hinman [1999], *Recursion on abstract structures*, ***Handbook of computability theory*** (E. R. Griffor, editor), North-Holland, pp. 315–359.

W. A. Howard [1973], *Hereditarily majorizable functionals of finite type*, ***Metamathematical investigation of intuitionistic arithmetic and analysis*** (A. S. Troelstra, editor), Lecture Notes in Mathematics, vol. 344, Springer-Verlag, pp. 454–461.

J. M. E. Hyland [1975], *Recursion theory on the countable functionals*, ***Ph.D. thesis***, University of Oxford.

J. M. E. Hyland [1978], *The intrinsic recursion theory on the countable or continuous functionals*, ***Generalized Recursion Theory II*** (J. E. Fenstad, R. O. Gandy, and G. E. Sacks, editors), North-Holland, pp. 135–145.

J. M. E. Hyland [1979], *Filter spaces and continuous functionals*, ***Ann. Math. Logic***, vol. 16, pp. 101–143.

J. M. E. Hyland [1982], *The effective topos*, ***The L. E. J. Brouwer Centenary Symposium*** (A. S. Troelstra and D. van Dalen, editors), North-Holland, pp. 165–216.

J. M. E. Hyland [1990], *First steps in synthetic domain theory*, ***Category Theory, Proceedings, Como*** (A. Carboni, M. C. Pedicchio, and G. Rosolini, editors), Lecture Notes in Mathematics, vol. 1488, Springer-Verlag, pp. 131–156.

J. M. E. Hyland [2002], *Variations on realizability: realizing the propositional axiom of choice*, ***Mathematical Structures in Computer Science***, vol. 12, no. 3, pp. 295–318.

J. M. E. Hyland, P. T. Johnstone, and A. M. Pitts [1980], *Tripos theory*, ***Mathematical Proceedings of the Cambridge Philosophical Society***, vol. 88, pp. 205–232.

J. M. E. Hyland and C.-H. L. Ong [2000], *On full abstraction for PCF: I, II and III*, ***Information and Computation***, vol. 163, pp. 285–408.

J. M. E. Hyland and A. Schalk [2002], *Games on graphs and sequentially realizable functionals*, ***Proceedings of 17th Annual Symposium on Logic in Computer Science***, IEEE, pp. 257–264.

R. Irwin, B. Kapron, and J. Royer [2001a], *On characterizations of the basic feasible functionals, Part I*, ***Journal of Functional Programming***, vol. 11, pp. 117–153.

R. Irwin, B. Kapron, and J. Royer [2001b], *On characterizations of the basic feasible functionals, Part II*, to appear.

A. JUNG AND A. STOUGHTON [1993], *Studying the fully abstract model of PCF within its continuous function model*, ***Proceedings of International Conference on Typed Lambda Calculi and Applications, Utrecht***, Lecture Notes in Computer Science, no. 664, Springer-Verlag, pp. 230–245.

G. KAHN AND G. D. PLOTKIN [1993], *Concrete domains*, ***Theoretical Computer Science***, vol. 121, pp. 187–277, first appeared in French as INRIA-LABORIA ***Technical report***, 1978.

R. KANNEGANTI, R. CARTWRIGHT, AND M. FELLEISEN [1993], *SPCF: its model, calculus, and computational power*, ***Proceedings of REX Workshop on Semantics and Concurrency***, Lecture Notes in Computer Science, vol. 666, Springer-Verlag, pp. 318–347.

A. S. KECHRIS [1973], *The structure of envelopes: A survey of recursion theory in higher types*, MIT Logic Seminar notes.

A. S. KECHRIS AND Y. N. MOSCHOVAKIS [1977], *Recursion in higher types*, ***Handbook of mathematical logic*** (J. Barwise, editor), North-Holland, pp. 681–737.

D. P. KIERSTEAD [1980], *A semantics for Kleene's j-expressions*, ***The Kleene Symposium*** (J. Barwise, H. J. Keisler, and K. Kunen, editors), North-Holland, pp. 353–366.

D. P. KIERSTEAD [1983], *Syntax and semantics in higher-type recursion theory*, ***Transactions of the American Mathematical Society***, vol. 276, pp. 67–105.

S. C. KLEENE [1936a], *General recursive functions of natural numbers*, ***Mathematische Annalen***, vol. 112, pp. 727–742.

S. C. KLEENE [1936b], *λ-definability and recursiveness*, ***Duke Mathematical Journal***, vol. 2, pp. 340–353.

S. C. KLEENE [1945], *On the interpretation of intuitionistic number theory*, ***The Journal of Symbolic Logic***, vol. 10, pp. 109–124.

S. C. KLEENE [1952], ***Introduction to metamathematics***, Wolter-Noordhoff and North-Holland.

S. C. KLEENE [1955a], *Arithmetical predicates and function quantifiers*, ***Transactions of the American Mathematical Society***, vol. 79, pp. 312–340.

S. C. KLEENE [1955b], *Hierarchies of number-theoretic predicates*, ***Bulletin of the American Mathematical Society***, vol. 61, pp. 193–213.

S. C. KLEENE [1959a], *Countable functionals*, ***Constructivity in Mathematics: Proceedings of the colloquium held at Amsterdam, 1957*** (A. Heyting, editor), North-Holland, pp. 81–100.

S. C. KLEENE [1959b], *Recursive functionals and quantifiers of finite types I*, ***Transactions of the American Mathematical Society***, vol. 91, pp. 1–52.

S. C. KLEENE [1962a], *Herbrand-Gödel-style recursive functionals of finite types*, ***Recursive function theory: Proceedings of Symposia on Pure Mathematics***, vol. 5, AMS, pp. 49–75.

S. C. KLEENE [1962b], *Lambda-definable functionals of finite types*, ***Fundamenta Mathematicae***, vol. 50, pp. 281–303.

S. C. KLEENE [1962c], *Turing-machine computable functionals of finite types I*, ***Logic, methodology and philosophy of science, Stanford***, pp. 38–45.

S. C. KLEENE [1962d], *Turing-machine computable functionals of finite types II*, ***Proceedings of the London Mathematical Society***, vol. 12, pp. 245–258.

S. C. KLEENE [1963], *Recursive functionals and quantifiers of finite types II*, ***Transactions of the American Mathematical Society***, vol. 108, pp. 106–142.

S. C. KLEENE [1969], ***Formalized recursive functionals and formalized realizability***, Memoirs of the American Mathematical Society, vol. 89, American Mathematical Society.

S. C. KLEENE [1978], *Recursive functionals and quantifiers of finite types revisited I*, ***Generalized Recursion Theory II*** (J. E. Fenstad, R. O. Gandy, and G. E. Sacks, editors), North-Holland, pp. 185–222.

S. C. KLEENE [1980], *Recursive functionals and quantifiers of finite types revisited II*, ***The Kleene Symposium*** (J. Barwise, H. J. Keisler, and K. Kunen, editors), North-Holland, pp. 1–29.

S. C. Kleene [1982], *Recursive functionals and quantifiers of finite types revisited III*, ***Patras Logic Symposion*** (G. Metakides, editor), North-Holland, pp. 1–40.

S. C. Kleene [1985], *Unimonotone functions of finite types* (*Recursive functionals and quantifiers of finite types revisited IV*), ***Recursion Theory*** (A. Nerode and R. A. Shore, editors), Proceedings of Symposia in Pure Mathematics, vol. 42, pp. 119–138.

S. C. Kleene [1991], *Recursive functionals and quantifiers of finite types revisited V*, ***Transactions of the American Mathematical Society***, vol. 325, pp. 593–630.

S. C. Kleene and R. E. Vesley [1965], ***The foundations of intuitionistic mathematics***, North-Holland.

U. Kohlenbach [2002], *Foundational and mathematical uses of higher types*, ***Reflections on the foundations of mathematics*** (W. Sieg, R. Sommer, and C. Talcott, editors), Lecture Notes in Logic, vol. 15, A K Peters, pp. 92–116.

P. G. Kolaitis [1978], *On recursion in E and semi-Spector classes*, ***Cabal seminar 1976–77*** (A. S. Kechris and Y. N. Moschovakis, editors), Lecture Notes in Mathematics, vol. 689, Springer-Verlag, pp. 209–243.

P. G. Kolaitis [1979], *Recursion in a quantifier vs. elementary induction*, ***The Journal of Symbolic Logic***, vol. 44, pp. 235–259.

P. G. Kolaitis [1980], *Recursion and nonmonotone induction in a quantifier*, ***The Kleene Symposium*** (J. Barwise, H. J. Keisler, and K. Kunen, editors), North-Holland, pp. 367–389.

P. G. Kolaitis [1985], *Canonical forms and hierarchies in generalized recursion theory*, ***Proceedings of Symposia on Pure Mathematics***, vol. 42, AMS, pp. 139–170.

M. V. Korovina and O. V. Kudinov [2001], *Semantic characterisations of second-order computability over the real numbers*, ***Computer Science Logic***, Lecture Notes in Computer Science, vol. 2142, Springer-Verlag, pp. 160–173.

G. Kreisel [1958], *Constructive mathematics*, notes of a course given at Stanford University.

G. Kreisel [1959], *Interpretation of analysis by means of functionals of finite type*, ***Constructivity in Mathematics: Proceedings of the colloquium held at Amsterdam, 1957*** (A. Heyting, editor), North-Holland, pp. 101–128.

G. Kreisel [1962], *On weak completeness of intuitionistic predicate logic*, ***The Journal of Symbolic Logic***, vol. 27, pp. 139–158.

G. Kreisel, D. Lacombe, and J. R. Shoenfield [1957], *Fonctionnelles récursivement définissable et fonctionnelles récursives*, ***Comptes Rendus de l'Académie des Sciences, Paris***, vol. 245, pp. 399–402.

G. Kreisel, D. Lacombe, and J. R. Shoenfield [1959], *Partial recursive functionals and effective operations*, ***Constructivity in Mathematics: Proceedings of the colloquium held at Amsterdam, 1957*** (A. Heyting, editor), North-Holland, pp. 101–128.

L. Kristiansen and D. Normann [1997], *Total objects in inductively defined types*, ***Archive of Mathematical Logic***, vol. 36, pp. 405–436.

C. Kuratowski [1952], ***Topologie***, vol. I, Warsaw.

A. H. Lachlan [1964], *Effective operations in a general setting*, ***The Journal of Symbolic Logic***, vol. 29, pp. 163–178.

D. Lacombe [1955a], *Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles I*, ***Comptes Rendus de l'Académie des Sciences, Paris***, vol. 240, pp. 2478–2480.

D. Lacombe [1955b], *Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles II*, ***Comptes Rendus de l'Académie des Sciences, Paris***, vol. 241, pp. 13–14.

D. Lacombe [1955c], *Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles III*, ***Comptes Rendus de l'Académie des Sciences, Paris***, vol. 241, pp. 151–153.

D. Lacombe [1955d], *Remarques sur les opérateurs récursifs et sur les fonctions récursives d'une variable réelle*, ***Comptes Rendus de l'Académie des Sciences, Paris***, vol. 241, pp. 1250–1252.

J. LAIRD [1997], *Full abstraction for functional languages with control*, ***Proceedings of 12th Annual Symposium on Logic in Computer Science***, IEEE, pp. 58–67.

J. LAIRD [1998], *A semantic analysis of control*, ***Ph.D. thesis***, University of Edinburgh, examined March 1999.

J. LAIRD [2002], *Games and sequential algorithms*, submitted.

J. LAMBEK AND P. J. SCOTT [1986], ***Introduction to higher-order categorical logic***, Cambridge Studies in Advanced Mathematics, vol. 7, Cambridge University Press.

B. LICHTENTHÄLER [1996], *Degrees of parallelism*, ***Technical Report 96-01***, Fachgruppe Informatik, Siegen.

R. LOADER [1997], *Equational theories for inductive types*, ***Annals of Pure and Applied Logic***, vol. 84, pp. 175–217.

R. LOADER [1998], *Unary PCF is decidable*, ***Theoretical Computer Science***, vol. 206, pp. 317–329.

R. LOADER [2001], *Finitary PCF is not decidable*, ***Theoretical Computer Science***, vol. 266, pp. 341–364.

M. H. LÖB [1970], *A model-theoretic characterization of effective operations*, ***The Journal of Symbolic Logic***, vol. 35, pp. 217–222, *a correction*, ibid., vol. 39, p. 225, 1974.

J. R. LONGLEY [1995], *Realizability toposes and language semantics*, ***Ph.D. thesis***, University of Edinburgh, available as ECS-LFCS-95-332.

J. R. LONGLEY [1999a], *Matching typed and untyped realizability*, ***Proceedings of Workshop on Realizability, Trento*** (L. Birkedal, J. van Oosten, G. Rosolini, and D. S. Scott, editors), published as Electronic Notes in Theoretical Computer Science 23 No. 1, Elsevier. Available via `http://www.elsevier.nl/locate/entcs/volume23.html`.

J. R. LONGLEY [1999b], *Unifying typed and untyped realizability*, unpublished note, available at `http://www.dcs.ed.ac.uk/home/jrl/unifying.txt`.

J. R. LONGLEY [1999c], *When is a functional program not a functional program?*, ***Proceedings of 4th International Conference on Functional Programming, Paris***, ACM Press, pp. 1–7.

J. R. LONGLEY [2002a], *The sequentially realizable functionals*, ***Annals of Pure and Applied Logic***, vol. 117, no. 1, pp. 1–93.

J. R. LONGLEY [2002b], *Universal types and what they are good for*, to appear in ***Proceedings of 2nd International Symposium on Domain Theory***, Chengdu.

J. R. LONGLEY AND G. D. PLOTKIN [1997], *Logical full abstraction and PCF*, ***Tbilisi Symposium on Language, Logic and Computation*** (J. Ginzburg et al., editor), SiLLI/CSLI, pp. 333–352.

J. R. LONGLEY AND A. K. SIMPSON [1997], *A uniform approach to domain theory in realizability models*, ***Mathematical Structures in Computer Science***, vol. 7, pp. 469–505.

G. LONGO AND E. MOGGI [1984a], *Cartesian closed categories of enumerations for effective type structures, Parts I and II*, ***Semantics of Data Types*** (G. Kahn, D. MacQueen, and G. Plotkin, editors), Springer-Verlag, pp. 235–255.

G. LONGO AND E. MOGGI [1984b], *The hereditary partial functionals and recursion theory in higher types*, ***The Journal of Symbolic Logic***, vol. 49, pp. 1319–1332.

F. LOWENTHAL [1976], *Equivalence of some definitions of recursion in a higher type object*, ***The Journal of Symbolic Logic***, vol. 41, pp. 427–435.

D. MACQUEEN [1972], *Post's problem for recursion in higher types*, ***Ph.D. thesis***, MIT, Cambridge, Massachusetts.

M. MARZ, A. ROHR, AND T. STREICHER [1999], *Full abstraction and universality via realisability*, ***Proceedings of 14th Annual Symposium on Logic in Computer Science***, IEEE, pp. 174–182.

S. MAZUR [1963], ***Computable analysis***, Rozprawy Matematyczne, vol. 33, Warsaw.

D. C. MCCARTY [1984], *Information systems, continuity and realizability*, ***Logics of Programs*** (E. Clarke and D. Cozen, editors), Lecture Notes in Computer Science, no. 164, Springer-Verlag, pp. 341–359.

R. MILNER [1977], *Fully abstract models of typed λ-calculi*, ***Theoretical Computer Science***, vol. 4, pp. 1–22.

R. MILNER, M. TOFTE, R. HARPER, AND D. MACQUEEN [1997], ***The definition of Standard ML (revised)***, MIT Press.

E. MOGGI [1988], *Partial morphisms in categories of effective objects*, ***Information and Computation***, vol. 73, pp. 250–277.

J. MOLDESTAD [1977], ***Computations in higher types***, Lecture Notes in Mathematics, vol. 574, Springer-Verlag.

Y. N. MOSCHOVAKIS [1967], *Hyperanalytic predicates*, ***Transactions of the American Mathematical Society***, vol. 129, pp. 249–282.

Y. N. MOSCHOVAKIS [1969], *Abstract first order computability I, II*, ***Transactions of the American Mathematical Society***, vol. 138, pp. 427–464, 465–504.

Y. N. MOSCHOVAKIS [1974a], ***Elementary induction on abstract structures***, North-Holland.

Y. N. MOSCHOVAKIS [1974b], *On non-monotone inductive definability*, ***Fundamenta Mathematicae***, vol. 82, pp. 39–83.

Y. N. MOSCHOVAKIS [1974c], *Structural characterizations of classes of relations*, ***Generalized Recursion Theory*** (J. E. Fenstad and P. G. Hinman, editors), North-Holland, pp. 53–79.

Y. N. MOSCHOVAKIS [1976], *On the basic notions in the theory of induction*, ***Proceedings of 5th International Congress in Logic, Methodology and Philosophy of Science***, London, Ontario, pp. 207–236.

Y. N. MOSCHOVAKIS [1981], *On the Grilliot-Harrington-MacQueen theorem*, ***Logic Year 1979–80***, Lecture Notes in Mathematics, vol. 859, Springer-Verlag, pp. 246–267.

Y. N. MOSCHOVAKIS [1983], *Abstract recursion as a foundation for the theory of algorithms*, ***Computation and Proof Theory: Proceedings of the Logic Colloquium, vol. 2*** (E. Boerger, W. Oberschelp, M. M. Richter, B. Schinzel, and W.Thomas, editors), Lecture Notes in Mathematics, vol. 1104, Springer-Verlag, pp. 289–364.

Y. N. MOSCHOVAKIS [1989], *The formal language of recursion*, ***The Journal of Symbolic Logic***, vol. 54, pp. 1216–1252.

K. MULMULEY [1987], ***Full abstraction and semantic equivalence***, MIT Press.

P. S. MULRY [1982], *Generalized Banach-Mazur functionals in the topos of recursive sets*, ***Journal of Pure and Applied Algebra***, vol. 26, pp. 71–83.

J. MYHILL AND J. C. SHEPHERDSON [1955], *Effective operations on partial recursive functions*, ***Zeitschrift für mathematische Logik und Grundlagen der Mathematik***, vol. 1, pp. 310–317.

A. NERODE [1957], *General topology and partial recursive functionals*, ***Cornell Summer Institute of Symbolic Logic***, Cornell, pp. 247–251.

A. NERODE [1959], *Some Stone spaces and recursion theory*, ***Duke Mathematical Journal***, vol. 26, pp. 397–406.

H. NICKAU [1994], *Hereditarily sequential functionals*, ***Proceedings of 3rd Symposium on Logical Foundations of Computer Science***, Lecture Notes in Computer Science, vol. 813, Springer-Verlag, pp. 253–264.

K.-H. NIGGL [1993], *Subrecursive hierarchies on Scott domains*, ***Archive of Mathematical Logic***, vol. 32, pp. 239–257.

K.-H. NIGGL [1999], *M^{ω} considered as a programming language*, ***Annals of Pure and Applied Logic***, vol. 99, pp. 73–92.

D. NORMANN [1978a], *A continuous functional with noncollapsing hierarchy*, ***The Journal of Symbolic Logic***, vol. 43, pp. 487–491.

D. NORMANN [1978b], *Set recursion*, ***Generalized Recursion Theory II*** (J. E. Fenstad, R. O. Gandy, and G. E. Sacks, editors), North-Holland, pp. 303–320.

D. NORMANN [1979a], *A classification of higher type functionals*, ***Proceedings of 5th Scandinavian Logic Symposium*** (F. V. Jensen, B. H. Mayoh, and K. K.Møller, editors), Aalborg University Press, pp. 301–308.

D. NORMANN [1979b], *Nonobtainable continuous functionals*, ***Proceedings of 6th International Congress on Logic, Methodology and Philosophy of Science***, Hanover, pp. 241–249.

D. NORMANN [1980], ***Recursion on the countable functionals***, Lecture Notes in Mathematics, vol. 811, Springer-Verlag.

D. NORMANN [1981a], *The continuous functionals: computations, recursions and degrees*, ***Annals of Mathematical Logic***, vol. 21, pp. 1–26.

D. NORMANN [1981b], *Countable functionals and the projective hierarchy*, ***The Journal of Symbolic Logic***, vol. 46, pp. 209–215.

D. NORMANN [1982], *External and internal algorithms on the continuous functionals*, ***Patras Logic Symposion*** (G. Metakides, editor), North-Holland, pp. 137–144.

D. NORMANN [1983], *Characterising the continuous functionals*, ***The Journal of Symbolic Logic***, vol. 48, pp. 965–969.

D. NORMANN [1989], *Kleene–spaces*, ***Logic colloquium 1988*** (Ferro, Bonotto, Valentini, and Zanardo, editors), Elsevier, pp. 91–109.

D. NORMANN [1997], *Closing the gap between the continuous functionals and recursion in* ^3E, ***Archive of Mathematical Logic***, vol. 36, pp. 269–287.

D. NORMANN [1999a], *The continuous functionals*, ***Handbook of computability theory*** (E. R. Griffor, editor), North-Holland, pp. 251–275.

D. NORMANN [1999b], *A Mahlo-universe of effective domains with totality*, ***Models and Computability*** (S. B. Cooper and J. K. Truss, editors), Cambridge University Press.

D. NORMANN [2000], *Computability over the partial continuous functionals*, ***The Journal of Symbolic Logic***, vol. 65, pp. 1133–1142.

D. NORMANN [2001], *The continuous functionals of finite types over the reals*, ***Domains and processes: Proceedings of 1st international symposium on domain theory, Shanghai*** (K. Keimel, G. Q. Zhang, Y. Liu, and Y. Chen, editors), Kluwer, pp. 103–124.

D. NORMANN [2002], *Exact real number computations relative to hereditarily total functionals*, to appear in ***Theoretical Computer Science***.

D. NORMANN AND C. RØRDAM [2002], *The computational power of* M^{ω}, ***Mathematical Logic Quarterly***, vol. 48, no. 1, pp. 117–124.

D. NORMANN AND S. S. WAINER [1980], *The 1-section of a countable functional*, ***The Journal of Symbolic Logic***, vol. 45, pp. 549–562.

P. G. ODIFREDDI [1989], ***Classical recursion theory, volume I***, Studies in Logic and the Foundations of Mathematics, vol. 125, Elsevier, second edition 1999.

P. W. O'HEARN AND J. G. RIECKE [1995], *Kripke logical relations and PCF*, ***Information and Computation***, vol. 120, no. 1, pp. 107–116.

C.-H. L. ONG [1995], *Correspondence between operational and denotational semantics*, ***Handbook of logic in computer science*** (S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors), vol. 4, Oxford University Press, pp. 269–356.

C.-H. L. ONG AND C. A. STEWART [1997], *A Curry-Howard foundation for functional computation with control*, ***Proceedings of 24th Symposium on Principles of Programming Languages***, ACM Press, pp. 215–227.

J. VAN OOSTEN [1991], *Exercises in realizability*, ***Ph.D. thesis***, University of Amsterdam.

J. VAN OOSTEN [1997], *The modified realizability topos*, ***Journal of Pure and Applied Algebra***, vol. 116, pp. 273–289.

J. VAN OOSTEN [1999], *A combinatory algebra for sequential functionals of finite type*, ***Models and Computability*** (S. B. Cooper and J. K. Truss, editors), Cambridge University Press, pp. 389–406.

J. VAN OOSTEN [2002], *Realizability: a historical essay*, ***Mathematical Structures in Computer Science***, vol. 12, no. 3, pp. 239–264.

P. PÄPPINGHAUS [1985], *Ptykes in Gödel's T und verallgemeinerte Rekursion über Mengen und Ordinalzahlen*, ***Habilitationsschrift***, Hannover.

R. Péter [1951a], *Probleme der Hilbertschen Theorie der höheren Stufen von rekursiven Funktionen*, ***Acta mathematica Academiae Scientarum Hungaricae***, vol. 2, pp. 247–274.

R. Péter [1951b], ***Rekursive Funktionen***, Akademischer Verlag, Budapest, English translation published as ***Recursive Functions***, Academic Press, 1967.

W. K.-S. Phoa [1991], *From term models to domains*, ***Proceedings of Theoretical Aspects of Computer Software, Sendai***, Lecture Notes in Computer Science, vol. 526, Springer-Verlag.

R. Platek [1966], *Foundations of recursion theory*, ***Ph.D. thesis***, Stanford University.

R. Platek [1971], *A countable hierarchy for the superjump*, ***Logic Colloquium 1969*** (R. O. Gandy and C. E. M. Yates, editors), North-Holland, pp. 257–271.

G. D. Plotkin [1977], *LCF considered as a programming language*, ***Theoretical Computer Science***, vol. 5, pp. 223–255.

G. D. Plotkin [1978], $\mathbb{T}^\omega$ *as a universal domain*, ***Journal of Computer and System Sciences***, vol. 17, pp. 209–236.

G. D. Plotkin [1983], *Domains*, ***Technical report***, Department of Computer Science, University of Edinburgh.

G. D. Plotkin [1997], *Full abstraction, totality and PCF*, ***Mathematical Structures in Computer Science***, vol. 9, no. 1, pp. 1–20.

E. L. Post [1936], *Finite combinatory processes—formulation 1*, ***The Journal of Symbolic Logic***, vol. 1, pp. 103–105.

M. B. Pour-El [1960], *A comparison of five "computable" operators*, ***Zeitschrift für mathematische Logik und Grundlagen der Mathematik***, vol. 6, pp. 325–340.

M. B. Pour-El [1999], *The structure of computability*, ***Handbook of computability theory*** (E. R. Griffor, editor), North-Holland, pp. 315–359.

M. B. Pour-El and J. I. Richards [1989], ***Computability in analysis and physics***, Springer-Verlag.

H. G. Rice [1953], *Classes of recursively enumerable sets and their decision problems*, ***Transactions of the American Mathematical Society***, vol. 74, pp. 358–366.

H. G. Rice [1956], *On completely recursively enumerable classes and their key arrays*, ***The Journal of Symbolic Logic***, vol. 21, pp. 304–308.

W. Richter [1967], *Constructive transfinite number classes*, ***Bulletin of the American Mathematical Society***, vol. 73, pp. 261–265.

J. G. Riecke [1993], *Fully abstract translations between functional languages*, ***Mathematical Structures in Computer Science***, vol. 3, pp. 387–415.

H. Rogers [1967], ***Theory of recursive functions and effective computability***, McGraw-Hill.

A. Rohr [2002], *A universal realizability model of sequential functional computation*, ***Ph.D. thesis***, Technical University of Darmstadt.

G. Rosolini [1986], *Continuity and effectiveness in topoi*, ***Ph.D. thesis***, Oxford; Carnegie-Mellon.

J. S. Royer [2000], *On the computational complexity of Longley's H functional*, presented at Second International Workshop on Implicit Computational Complexity, UC/Santa Barbara.

G. E. Sacks [1971], *Recursion in objects of finite type*, ***Proceedings of International Congress of Mathematicians***, Gauthiers-Villars, Paris, pp. 251–254.

G. E. Sacks [1974], *The 1-section of a type n object*, ***Generalized Recursion Theory*** (J. E. Fenstad and P. G. Hinman, editors), North-Holland, pp. 81–96.

G. E. Sacks [1977], *The k-section of a type n object*, ***American Journal of Mathematics***, vol. 99, pp. 901–917.

G. E. Sacks [1980], *Post's problem, absoluteness and recursion in finite types*, ***The Kleene Symposium*** (J. Barwise, H. J. Keisler, and K. Kunen, editors), North-Holland, pp. 201–222.

G. E. Sacks [1985], *Post's problem in E-recursion*, ***Proceedings of Symposia on Pure Mathematics***, vol. 42, AMS, pp. 177–193.

G. E. Sacks [1986], *On the limits of E-recursive enumerability*, ***Annals of Pure and Applied Logic***, vol. 31, pp. 87–120.

G. E. Sacks [1990], ***Higher recursion theory***, Springer-Verlag.

G. E. Sacks [1999], *E-recursion*, ***Handbook of computability theory*** (E. R. Griffor, editor), Elsevier, pp. 301–314.

L. E. Sanchis [1967], *Functionals defined by recursion*, ***Notre Dame Journal of Formal Logic***, vol. 8, pp. 161–174.

L. E. Sanchis [1992], ***Recursive functionals***, Studies in Logic and the Foundations of Mathematics, vol. 131, North-Holland.

L. P. Sasso [1971], *Degrees of unsolvability of partial functions*, ***Ph.D. thesis***, University of California, Berkeley.

V.Yu. Sazonov [1975], *Sequentially and parallelly computable functionals*, ***λ-calculus and Computer Science Theory: Proceedings of the symposium held in Rome***, Lecture Notes in Computer Science, vol. 37, Springer-Verlag, pp. 312–318.

V.Yu. Sazonov [1976a], *Degrees of parallelism in computations*, ***Proceedings of Symposium on Mathematical Foundations of Computer Science***, Lecture Notes in Computer Science, vol. 45, Springer-Verlag, pp. 517–523.

V.Yu. Sazonov [1976b], *Expressibility of functions in Scott's LCF language*, ***Algebra i Logika***, vol. 15, pp. 308–330.

V.Yu. Sazonov [1976c], *Functionals computable in series and in parallel*, ***Matematicheskii Zhurnal***, vol. 17, pp. 648–672.

V.Yu. Sazonov [1998], *An inductive definition of full abstract model for LCF (preliminary version)*, available from `http://csc.liv.ac.uk`.

B. Scarpellini [1971], *A model for barrecursion of higher types*, ***Compositio Mathematica***, vol. 23, pp. 123–153.

H. Schwichtenberg [1975], *Elimination of higher type levels in definitions of primitive recursive functionals by means of transfinite recursion*, ***Logic Colloquium 1973*** (H. Rose and T. Shepherdson, editors), North-Holland, pp. 279–303.

H. Schwichtenberg [1991], *Primitive recursion on the partial continuous functionals*, ***Informatik und Mathematik*** (M. Broy, editor), Springer-Verlag, pp. 251–269.

H. Schwichtenberg [1996], *Density and choice for total continuous functionals*, ***Kreiseliana. About and around Georg Kreisel*** (P. Odifreddi, editor), A. K. Peters, Wellesley, Massachusetts, pp. 335–362.

H. Schwichtenberg [1999], *Classifying recursive functions*, ***Handbook of computability theory*** (E. R. Griffor, editor), North-Holland, pp. 533–586.

D. S. Scott [1969], *A theory of computable functions of higher type*, unpublished seminar notes, University of Oxford. 7 pages.

D. S. Scott [1970], *Outline of a mathematical theory of computation*, ***Proceedings of 4th Annual Princeton Conference on Information Science and Systems***, pp. 165–176.

D. S. Scott [1972], *Continuous lattices*, ***Toposes, Algebraic Geometry and Logic*** (F. W. Lawvere, editor), Springer-Verlag.

D. S. Scott [1976], *Data types as lattices*, ***SIAM Journal of Computing***, vol. 5, no. 3, pp. 522–587.

D. S. Scott [1982], *Domains for denotational semantics*, ***Proceedings of 9th International Colloquium on Automata, Languages and Programming, Aarhus, Denmark*** (M. Nielsen and E. M. Schmidt, editors), Lecture Notes in Computer Science, no. 140, Springer-Verlag, pp. 577–610.

D. S. Scott [1993], *A type-theoretical alternative to ISWIM, CUCH, OWHY*, ***Theoretical Computer Science***, vol. 121, pp. 411–440, first written in 1969 and widely circulated in unpublished form since then.

J. R. Shoenfield [1962], *The form of the negation of a predicate*, ***Recursive function theory: Proceedings of Symposia on Pure Mathematics*** (J. C. E. Dekker, editor), vol. 5, AMS, pp. 131–134.

J. R. Shoenfield [1967], ***Mathematical logic***, Addison-Wesley.

J. R. Shoenfield [1968], *A hierarchy based on a type two object*, ***Transactions of the American Mathematical Society***, vol. 134, pp. 103–108.

K. Sieber [1990], *Relating full abstraction results for different programming languages*, ***Proceedings of 10th Conference on Foundations of Software Technology and Theoretical Computer Science, Bangalore***, Lecture Notes in Computer Science, vol. 472, Springer-Verlag.

K. Sieber [1992], *Reasoning about sequential functions*, ***Applications of Categories in Computer Science, Durham 1991*** (M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors), London Mathematical Society Lecture Note Series, vol. 177, Cambridge University Press, pp. 258–269.

A. Simpson [1998], *Lazy functional algorithms for exact real functionals*, ***Mathematical Foundations of Computer Science, Proceedings***, Lecture Notes in Computer Science, vol. 1450, Springer-Verlag, pp. 456–464.

T. A. Slaman [1981], *Aspects of E-recursion*, ***Ph.D. thesis***, Harvard University.

T. A. Slaman [1985], *The E-recursively enumerable degrees are dense*, ***Proceedings of Symposia on Pure Mathematics***, vol. 42, AMS, pp. 195–213.

C. Spector [1962], *Provably recursive functionals of analysis: a consistency proof of analysis by means of principles formulated in current intuitionistic mathematics*, ***Recursive function theory: Proceedings of Symposia on Pure Mathematics***, vol. 5, AMS, pp. 1–27.

D. Spreen [1992], *Effective operators and continuity revisited*, ***Logical Foundations of Computer Science, Second International Symposium, Tver*** (A. Nerode and M. A. Taitslin, editors), Lecture Notes in Computer Science, vol. 620, Springer-Verlag, pp. 459–469.

D. Spreen and P. Young [1983], *Effective operators in a topological setting*, ***Computation and Proof Theory: Proceedings of the Logic Colloquium, vol. 2*** (E. Boerger, W. Oberschelp, M. M. Richter, B. Schinzel, and W.Thomas, editors), Lecture Notes in Mathematics, vol. 1104, Springer-Verlag, pp. 437–451.

V. Stoltenberg-Hansen, I. Lindström, and E. R. Griffor [1994], ***Mathematical theory of domains***, Cambridge Tracts in Theoretical Computer Science, no. 22, Cambridge University Press.

A. Stoughton [1991a], *Interdefinability of parallel operations in PCF*, ***Theoretical Computer Science***, vol. 79, pp. 357–358.

A. Stoughton [1991b], *Parallel PCF has a unique extensional model*, ***Proceedings of 6th Annual Symposium on Logic in Computer Science***, IEEE, pp. 146–151.

W. W. Tait [1962], *A second order theory of functionals of higher type*, in Stanford report on the foundations of analysis, Stanford University.

W. W. Tait [1967], *Intensional interpretations of functionals of finite type I*, ***The Journal of Symbolic Logic***, vol. 32, pp. 198–212.

M. B. Trakhtenbrot [1975], *On representation of sequential and parallel functions*, ***Proceedings of 4th Symposium on Mathematical Foundations of Computer Science***, Lecture Notes in Computer Science, vol. 32, Springer-Verlag, pp. 411–417.

A. S. Troelstra [1973], ***Metamathematical investigation of intuitionistic arithmetic and analysis***, Lecture Notes in Mathematics, vol. 344, Springer-Verlag, second edition (with corrections): ILLC Prepublication Series X-93-05, University of Amsterdam, 1993.

G. S. Tseitin [1959], *Algorithmic operators in constructive complete separable metric spaces*, ***Doklady Akademii Nauk***, vol. 128, pp. 49–52.

G. S. Tseitin [1962], *Algorithmic operators in constructive metric spaces*, ***Trudy Matematicheskogo Instituta imeni V. A. Steklova***, vol. 67, pp. 295–361, translated in AMS Translations (2), vol. 64 (1966), 1–80.

J. V. Tucker [1980], *Computing in algebraic systems*, ***Recursion theory: its generalizations and applications*** (F. R. Drake and S. S. Wainer, editors), London Mathematical Society Lecture Notes, vol. 45, Cambridge University Press, pp. 215–235.

T. Tugué [1960], *Predicates recursive in a type-2 object and Kleene hierarchies*, ***Commentarii Mathematici Universitatis Sancti Pauli, Tokyo***, vol. 8, pp. 97–117.

A. M. Turing [1937a], *Computability and λ-definability*, ***The Journal of Symbolic Logic***, vol. 2, pp. 153–163.

A. M. Turing [1937b], *On computable numbers, with an application to the Entscheidungsproblem*, ***Proceedings of the London Mathematical Society***, vol. 42, pp. 230–265, *a correction*, ibid., vol. 42, pp. 455–546, 1937.

A. M. Turing [1939], *Systems of logic based on ordinals*, ***Proceedings of the London Mathematical Society***, vol. 45, pp. 161–228.

V. A. Uspenskii [1955], *On enumeration operators*, ***Doklady Akademii Nauk***, vol. 103, pp. 773–776.

J. Vuillemin [1973], *Correct and optimal implementations of recursion in a simple programming language*, ***Proceedings of 5th ACM Symposium on Theory of Computing***, pp. 224–239.

S. S. Wainer [1974], *A hierarchy for the 1-section of any type two object*, ***The Journal of Symbolic Logic***, vol. 39, pp. 88–94.

S. S. Wainer [1975], *Some hierarchies based on higher type quantification*, ***Logic Colloquium 1973*** (H. Rose and T. Shepherdson, editors), North-Holland, pp. 305–316.

S. S. Wainer [1978], *The 1-section of a non-normal type-2 object*, ***Generalized Recursion Theory II*** (J. E. Fenstad, R. O. Gandy, and G. E. Sacks, editors), North-Holland, pp. 407–417.

K. Weihrauch [1985], *Type 2 recursion theory*, ***Theoretical Computer Science***, vol. 38, pp. 17–33.

K. Weihrauch [2000a], ***Computability***, EATCS Monographs on Theoretical Computer Science, vol. 9, Springer-Verlag.

K. Weihrauch [2000b], ***Computable analysis: an introduction***, Texts in Theoretical Computer Science, Springer-Verlag.

P. R. Young [1968], *An effective operator, continuous but not partial recursive*, ***Proceedings of the American Mathematical Society***, vol. 19, pp. 103–108.

P. R. Young [1969], *Toward a theory of enumerations*, ***Journal for the Association of Computing Machinery***, vol. 16, pp. 328–348.

I. D. Zaslavskii [1955], *Refutation of some theorems of classical analysis in constructive analysis*, ***Uspekhi Matematichekikh Nauk***, vol. 10, pp. 209–210.

I. D. Zaslavskii and G. S. Tseitin [1962], *On singular coverings and properties of constructive functions connected with them*, ***Trudy Matematicheskogo Instituta imeni V. A. Steklova***, vol. 67, pp. 458–502, translated in AMS Translations (2), vol. 98 (1971), 41–89.

J. I. Zucker [1971], *Proof theoretic studies of systems of iterated inductive definitions and subsystems of analysis*, ***Ph.D. thesis***, Stanford University.

INDEX

$[\cdots]$, $\langle\cdots\rangle$, 10
$[x_i]$, 39
$U[V/x]$, 13
X/R, 10
$\{m\}(\boldsymbol{x})$, 27
$\boldsymbol{x}$ (boldface), 11, 27
$\boldsymbol{x} : \boldsymbol{\sigma}$, 27
$[\![U]\!]$, 14, 15
$\lceil f \rceil$, 49
$\widetilde{f}$, 10
$\overline{n}$, 11
$\widehat{\theta}$, 20
:, 14, 27
$\to$, 11
$\cdot$, 12
$\cdot_{\sigma,\tau}$, 12

$\Downarrow$, 14
$|$, 36
$\rightharpoonup$, 9
$\preceq$, 29
$\preceq_c$, 41
$\rightsquigarrow$, 14
$\rightsquigarrow^*$, 14
$\sqsubseteq$, 53
$=, \simeq$, 10
$\#$, 27
$\bot, X_\bot$, 9, 46
$\downarrow$, 10, 39
$\uparrow$, 10, 54
$\mathtt{0}$, 23, 25
${}^2\exists$, 5, 29
${}^k\exists$, 32
$\Lambda x.e$, 10
$\lambda x.U$, 13
$\mu n.e$, 22
Φ, 38
ϕ_m, 10
C, 36
$\mathcal{C}_2, \mathcal{C}_{20}$, 55
D, 47
$\mathsf{EC}(-)$, 12
ECF, 36
$\mathsf{ECF}(\mathcal{R})$, 43
$\boldsymbol{EN}$, 55, 57
F_x, 54
HEO, 42
HRC, 43
HRO, 42
I_σ, 47
$K_{\sigma\tau}$, 47
$\mathsf{KC}(-)$, 30, 38
M, 46
$\mathbb{N}$, 10
$\mathbb{N}^\mathbb{N}, \mathbb{N}^\mathbb{N}_p, \mathbb{N}^\mathbb{N}_{rec}, \mathbb{N}^\mathbb{N}_{p\ rec}$, 10
$\mathcal{N}_\sigma$, 36
P, 53
$\boldsymbol{PER}(-)$, 44
$\mathsf{P}^{\mathit{eff}}$, 53, 54, 57
por, 48
$\mathcal{R}(-)$, 47
RC, 37
$\mathtt{rec}$, 23
$\widehat{\mathtt{rec}}$, 25
$S_{\rho\sigma\tau}$, 47
S, 12
Seq$(-)$, 10
${}^kS$, 35
$\mathtt{succ}$, 23, 25
$\mathsf{T}(-,-)$, 15
$\mathbb{T}^\omega$, 57
U, 51
$X(-)$, 27
Y_σ, 47
Z, 22

Aberth, 7, 21
Abramsky, 16
abstract computability, 5, 31, 34
Ackermann function, 18, 26
Aczel, 5, 35
adequacy, 49
admissible set theory, 5, 33, 35
algebraic
  DCPOs, 54
  lattices, 39
anti-Church's thesis
  for total functionals, 43
associates, 36, 42, 43

Banach-Mazur functional, 16
  generalized, 18, 57
bar recursion, 25, 39
Barendregt, 25, 26
basic types, 11
Bauer, 7, 41, 42
Beeson, 5, 7, 21
Bellantoni, 6
Berger, 16, 41
Bergstra, 39, 40
Berry, 16, 52, 53
β-rule, 14, 16
Bezem, 44
Birkedal, 41
black boxes, 20, 21, 26
Brouwer, 25, 38
Bucciarelli, 53

call-by-name (CBN), 45, 52, 55
call-by-value (CBV), 45, 46, 52, 55
cartesian closed category, 15
Church, 16, 29
Church's thesis, 1, 16
  at higher types, 8, 28, 50
Clarke, 35
codensity, 41
coherent domains, 57
compactly generated Hausdorff spaces, 39
complete f_0-spaces, 56
complete lattices, 54

complete partial orders, 53
complexity, 6
  logical, 5, 29, 34, 40
computation trees, 28, 50
congruence rules, 14, 16
consistent
  elements, 54
  functionals, 46
consistently complete, 54
constructibility, 5
constructive
  logic, 4, 24, 37
continuity, 19, 21, 44, 53
continuous
  functionals, 36
  functions, 53
Cook, 2, 6, 16
Cooper, 7
countable functionals, 36
course-of-values function, 10
CPOs, 53
Curien, 16, 52, 53

datatypes, 6
Davis, 46
DCPOs, 54
definability
  recursive, 47
($\mathcal{L}$-)definable, 14
definedness, 10
degrees, 33, 35, 40–41
  Kleene, 29, 40
density, 41
  theorem, 37, 41
dependent types, 41
descriptive set theory, 5
Dialectica
  interpretation, 23–25
  translation, 24, 37
directed
  sets, 54
domains, 54
double-negation translation, 24, 38
Draanen, van, 52
Dragalin, 24

E-recursion, 35
effective
  elements, 54
  enumeration, 54
  operation, 20

Ehrhard, 16
enumerated sets, 55
enumerators, 49
envelopes, 33–34, 41
equilogical spaces, 41
Ershov, 16, 39, 41, 44, 53, 55–57
Escardó, 7
extensional
  collapse, 12, 16, 36, 42, 44, 56
  type structure, 12

f-sets, 55
f_0-spaces, 55–57
fan functional, 38, 43
Feferman, 5, 26, 31, 40
Fenstad, 31, 34
filter spaces, 39
final value, 14
finite
  elements, 20, 54
  information, 36, 37, 54, 55
  types, 11
first recursion theorem, 29, 30, 48, 50
fixed point operators, 47
Friedberg, 18, 21
Friedman, 5
functionals, 2, 12

Gödel, 5, 6, 16, 22–26
game models
  of PCF, 52
Gandy, 16, 21, 24, 31, 32, 34, 35, 38, 40, 44, 48
Girard, 25, 44
Griffor, 35
Grilliot, 16, 33, 34, 39, 40
Grzegorczyk, 7, 16, 19, 21, 24
  hierarchy, 6

Harrington, 16, 33–35
Henkin, 16
Herbrand-Gödel equations, 19, 29, 48
hereditarily, 12
  effective operations, 42–44
  extensional, 56
  majorizable, 25
  monotone functionals, 46
  recursive operations, 42, 44
  recursively countable functionals, 43
  total, 13, 56
hierarchies, 34–35
Hilbert, 16

Hinata, 24, 37
Hinatani, 24
Hinman, 5, 16, 34, 35, 40
Howard, 25
Hyland, 38–41, 44, 50, 53
hyperanalytic hierarchy, 35
hyperarithmetic
  functions, 29, 34
  hierarchy, 34
hyperfinite type structure, 39

ideals, 56
index, 28
  application, 27
  recursive, 10, 20, 27
information systems, 55
interpretation, 14, 15
irreducible element, 40
Irwin, 7, 26

j-expression, 50

Kapron, 7, 26
Kechris, 31, 33
Kierstead, 16, 52
  functional, 51
Kleene, 5, 6, 10, 16, 18–19, 21, 22, 25–31, 34, 36–38, 45, 49–53
  computability, 27, 30
    over C, 38
    over S, 30–35, 48
    relative, 29
  degree, 29, 40
  equality, 10
  tree, 21, 43
Kohlenbach, 5
Korovina, 7
Kreisel, 5, 16, 22, 24, 25, 36–38, 42–43
Kreisel-Lacombe-Shoenfield theorem, 21
  generalized, 41, 43, 44
Kristiansen, 41
Kudinov, 7

L-spaces, 39
Lacombe, 7, 16, 21
λ-calculus, 1, 29
  simply typed, 13, 15
  untyped, 13, 48, 57
Lambek, 16
Lawvere, 18
level (of a type), 11
Loader, 16, 25
Longley, 6, 16
Longo, 57
Lowenthal, 34

MacQueen, 33
Markov, 7
  algorithms, 1
Martin-Löf, 41
model, 16
modulus
  of continuity, 19
  of extensionality, 25
  of uniform continuity, 38
Moggi, 57
Moldestad, 34, 49
monotone partial functionals, 46
Moschovakis, 5, 6, 16, 31–35
Mulry, 18, 57
Myhill, 16
Myhill-Shepherdson theorem, 20
  generalized, 56

neighbourhoods, 36
  formal, 37, 43
Nerode, 20
Niggl, 6, 7
no-counterexample interpretation, 38
non-termination, 9
normal form theorem, 29
normal objects/functionals, 31–35
Normann, 5, 7, 16, 35, 38–41
number selection, 32

object
  type *n*, 12
observational equivalence, 25
Odifreddi, 1, 21
Ong, 50, 53
operation
  effective, 20
oracles, 16, 19, 20, 26–29, 51–53

Péter, 16, 18–19, 22, 23, 26
parallel or, 48
partial continuous functionals, 53
partial equivalence relations, 10, 44
partial recursive
  uniformly, 19
PCF, 48
physics, 7
Platek, 16, 22, 31, 32, 35, 46–50, 52
Plotkin, 6, 16, 57

plus-one theorem, 33
plus-two theorem, 33
Post, 16
Post's problem, 33
Pour-El, 7, 18, 22
primitive recursion
  at higher types, 23, 26
  at type 2, 18
  uniform, 19
programming languages, 6
projective hierarchy, 40
pure types, 11
Putnam, 46

quantifiers, 5, 29

real numbers, 7
realizability, 4
recursive topos, 57
recursively
  continuous functionals, 37
  countable functionals, 37
reduction, 14
representations
  of sets, 41
represents, 47
retracts, 6, 57
Rice, 16, 20
Richards, 7
Richter, 34
Rogers, 18, 21
Royer, 7, 26

S1–S8, 25–26
S1–S9, 26–31, 48
  spirit of, 26
S8, anomalies arising from, 30–31, 44
S9, 28
S11, 50–52
Sacks, 5, 16, 33, 35
Sanchis, 24
Sazonov, 16
Scarpellini, 39
Schwichtenberg, 6, 7, 41
Scott, 6, 16, 39, 41, 53–55
  domains, 54
second recursion theorem, 21, 50
sections, 33, 40
selection theorems, 32–33
sequential
  algorithms, 52, 53
  continuity, 39, 44
  strategies, 51
set theory, 31
  admissible, 5, 33, 35
  descriptive, 5
Shapiro, 16, 20
Shepherdson, 16
Shoenfield, 16, 24, 34
simple types, 11
Simpson, 7
Slaman, 35
Spector, 25
stage comparison, 32
strong normalization, 24
strong or, 48
subrecursion, 6
superjump, 35
Suslin quantifier, 34
System T, 6, 23–26
System $\widehat{\mathrm{T}}$, 26

Tait, 16, 24, 37, 38, 43
term models, 14
theories, 14, 25
Θ-computability, 32
topos, 16
  recursive, 57
total
  continuous functionals, 25, 35–42
total recursive
  uniformly, 19
totality
  in domains, 41
transfinite
  iteration, 47
  types, 3, 16, 41
Troelstra, 5, 7, 16, 25, 43
Tseitin, 21
Tucker, 5
Tugué, 34, 37
Turing, 16
  machines, 1, 16, 29, 41
type structure, 12
  extensional, 12
  over X, 12
  partial, 12
  total, 12
type two effectivity, 41

uniformly (primitive, partial, total) recursive, 18
unimonotone functionals, 51

universal
  domain, 57
  type, 57
Uspenskii, 20

Wainer, 34, 40
Weihrauch, 7, 41
well-pointed categories, 15

Zaslavskii, 21
Zucker, 16

DIVISION OF INFORMATICS
UNIVERSITY OF EDINBURGH
EDINBURGH, EH9 3JZ, UK
E-mail: jrl@dcs.ed.ac.uk