FEFERMAN ON COMPUTABILITY

JEFFERY ZUCKER

To the memory of Sol Feferman, who, with his unfailing kindness, patience and good humour, was a constant source of inspiration to me, ever since his supervision of my graduate studies at Stanford

Abstract. Solomon Feferman has left his mark on computability theory, as on many other areas of foundational studies. The purpose of this chapter is, by means of reviewing a selected few of his many papers in this area, to give an idea of his impressive insights and developments in this field.

Introduction

Classical computability theory (or recursion theory) investigates the computability of functions on the domain of natural numbers $\mathbb{N}$, or (equivalently) strings over a finite alphabet. This study can be generalized in two directions: investigating (i) functionals of higher types over $\mathbb{N}$, and/or (ii) function(al)s over more general domains, such as the reals, with their distinctive topological and other properties. Over a period of about four decades, Sol Feferman (hereinafter SF) carried out highly significant investigations in both directions. The aim of this chapter is not to give a complete survey of his work in this area, which would be very difficult because of the profusion of his writings, but to examine a few of his more noteworthy papers spanning this period, so as to get a taste of his research here.

We will investigate four of SF's papers, spanning this period:

1. "Inductive Schemata and Recursively Continuous Functionals" (1977) [Fef77b],
2. "A New Approach to Abstract Data Types", Parts I and II (1992) [Fef92a, Fef92b],
3. "Computation on Abstract Data Types: The Extensional Approach, with an Application to Streams" (1996) [Fef96], and
4. "About and Around Computing over the Reals" (2013) [Fef13].

A section is devoted to each of these papers below.

It is hoped that the summaries given here will encourage researchers, students and historians to read the original papers. In such cases, one will be impressed not only by the contents, but by the clarity and elegance of SF's scientific literary style. For the same reason, as will be seen, I have frequently taken to quoting directly from these papers, since I often found that to be the best way to convey SF's ideas.[1]

Thanks to Wilfried Sieg and an anonymous referee for helpful comments on an earlier draft, and to the Natural Sciences and Engineering Research Council (Canada) for support.

[1] To avoid confusion, sections in SF's papers are referred to as Section 1, etc., and sections in the present article as §1, §2, etc.

1. Inductive Schemata and Recursively Continuous Functionals

The first paper to be reviewed here is [Fef77b]. Its topic is a particular approach to generalized recursion theory (g.r.t), based on *monotone inductive schemata over arbitrary structures*. It has 9 sections, and is moreover divided into two main parts.

Part I is an informal introduction to this theory. To quote SF: "the ideas ... are due independently to Moschovakis and the author, but the detailed development is due almost entirely to Moschovakis. A principal source for these ideas is in Platek's work on fixed-point schemata". This refers to Richard Platek's PhD thesis [Pla66] written under SF's supervision at Stanford[2].

Section 2 (still in Part I) gives a review of Platek's work, and how it was modified and expanded for the present purpose. As SF puts it, Platek's main idea was that the central feature of recursion theory which makes sense for *arbitrary structures* $\mathcal{A} = \langle A, \ldots \rangle$ is the process of recursion itself, i.e., the definition of a function φ as the *least fixed point* of an equation $\varphi = \Phi(\varphi)$; or, equivalently, $\mathbf{FP}(\Phi)$ is the least φ "closed under" Φ: $\Phi(\varphi) \subseteq \varphi$. For this to make sense, we must assume that the functional Φ is a *monotonic* operator on partial functions on A. Platek introduced a structure of *hereditarily monotonic functionals*[3] of finite type over A, denoted $\tilde{\mathcal{A}} = \langle \tilde{A}_\tau \rangle_\tau$, where τ ranges over all finite types.

Next, for any class $\mathcal{F}$ of functionals in $\tilde{\mathcal{A}}$, let $\mathbf{Ind}(\mathcal{F})$ be the class of functionals inductively defined from $\mathcal{F}$, obtained by closing under *explicit definition* and $\mathbf{FP}$ at all types.

Two of the main results of [Pla66] are:

- If every member of $\mathcal{F}$ has type level $\leq n+1$ and $\varphi \in \mathbf{Ind}(\mathcal{F})$ has type level $\leq n$ then we can get φ from $\mathcal{F}$ by explicit definition and the $\mathbf{FP}$ scheme at type levels $\leq n$ only. In particular, with the structure $\mathcal{A} = (A, \mathcal{F})$, if $\mathcal{F}$ has only total functions of level 1, we need only use Φ of type level 2.
- The system of functionals given by Kleene's schemata S1–S9 [Kle59b][4] on the *maximal type structure* over $\mathbb{N}$, i.e. $\mathcal{N} = (\mathbb{N}_n)_{n\in\mathbb{N}}$ (where $\mathbb{N}_0 = \mathbb{N}$ and $\mathbb{N}_{n+1} = \mathbb{N}^{\mathbb{N}_n}$), is equivalent to $\mathbf{Ind}(0, \boldsymbol{Sc})$[5], in the sense that there is an embedding of $\mathcal{N}$ into $\tilde{\mathcal{N}}$ such that Φ on $\mathcal{N}$ is Kleene partial recursive (i.e., derivable from S1–S9) iff its image in $\tilde{\mathcal{N}}$ is in $\mathbf{Ind}(0, \boldsymbol{Sc})$.

We should note that Platek makes some special assumptions on the structure $\langle A, \mathcal{F} \rangle$, namely that $\mathcal{F}$ (or $\mathbf{Ind}(\mathcal{F})$) contains pairing and projection functions, and distinct elements 0,1 of A. With these special assumptions on a level 1 structure $\langle A, \mathcal{F} \rangle$, the functions in $\mathbf{Ind}(\mathcal{F})$ turn out to be the same as the *prime computable* functions of Moschovakis [Mos69] on $\langle \mathcal{F}, =_A \rangle$.

SF then notes certain limitations on Platek's theory:

(1) These special assumptions effectively introduce $\mathcal{N}$ as a substructure of $\mathcal{A}$, and hence ordinary recursion theory as part of $\mathbf{Ind}(\mathcal{F})$. SF argues against this (in Section 4, as we will see).

---

[2] and regrettably never published, but with a far-ranging influence, as we will see in some of the other papers investigated here.

[3] This terminology is SF's.

[4] Kleene's notion of *partial recursive functional*, given by his schemata S1–S9 [Kle59b] or (equivalently for type levels ≤ 2) as in [Kle52, p. 326], will feature in every one of the four articles discussed here.

[5] where '$\boldsymbol{Sc}$' is the successor function on $\mathbb{N}$.

(2) It does not generalize recursion theories based on *relations*, such as Post-Smullyan systems [Smu61] or search computability [Mos69]).
(3) Recursion on $\mathcal{N}$ is derived indirectly, from embedding into $\tilde{\mathcal{N}}$, which is "messy". It would be preferable, conceptually, to *identify* the S1–S9 functionals over $\mathcal{N}$ with $\mathbf{Ind}(\mathcal{F})$ over the ground domain system $\langle \mathbb{N}_n \rangle_n$ for suitable $\mathcal{F}$, i.e., more generally, associate a suitable recursion theory with any ground structure $\langle \langle A_i \rangle_{i \in I}, \mathcal{F} \rangle$, for which recursion on $\langle \mathbb{N}_n \rangle_n$ would be one example.

Section 3 of this paper presents SF's general theory of monotone inductive definitions which overcomes these three limitations. SF found that there was a large overlap of his theory with that of Moschovakis, which had been developed independently (and also included a theory of non-monotone inductive definitions) [Mos76, KM77].

Briefly: Suppose given a *domain system* $\langle A_i \rangle_{i \in I}$ and a collection $\mathcal{X}$ of relations on these, closed under unions of chains. Assume each $X \in \mathcal{X}$ has an *arity* $\nu = (\nu(1), \ldots, \nu(m))$, with $X \subseteq A_{\nu(1)} \times \cdots \times A_{\nu(m)}$. A *monotone schema* is a functional $\Phi \colon \mathcal{X}_{\nu(1)} \times \cdots \times \mathcal{X}_{\nu(m)} \to \mathcal{X}_\mu$ which is monotonic, in the sense that

$$X_i \subseteq X_i' \ (1 \leq i \leq m) \implies \Phi(X_1, \ldots, X_m) \subseteq \Phi(X_1', \ldots, X_m').$$

With each such Φ we can associate a least fixed point $\mathbf{FP}(\Phi)$. Then, given a collection $\mathcal{F}$ of schemata, we take $\mathbf{Ind}(\mathcal{F})$ to be the smallest collection of schemata containing $\mathcal{F}$ and closed under explicit definition and $\mathbf{FP}$.

Given a domain system $\langle A_i \rangle_{i \in I}$, there are two principal choices for $\mathcal{X}$: (i) all relations on $\langle A_i \rangle_{i \in I}$, and (ii) all partial functions on $\langle A_i \rangle_{i \in I}$. These give rise to (respectively) *relational* and *functional* inductive theories. From these Moschovakis's *prime computability* and *search computability* [Mos69] can easily be constructed on an arbitrary A (or rather the closure of $A \cup \{0\}$ under pairing).

Another example is recursion on the *maximal type structure* $\mathcal{N} = \langle \mathbb{N}_n \rangle_n$ over $\mathbb{N}$. Here we take the class $\mathbf{Ind}(\mathcal{F})$ where $\mathcal{F}$ contains 0, successor, some basic functions and two schemata for application and abstraction at all types. Now we can derive, in $\mathbf{Ind}(\mathcal{F})$, Kleene's [Kle59b] schema S8 (higher order function abstraction, combined with application) and an "enumeration" schema S9:

$$\varphi(\vec{\alpha}, z) \simeq \{z\}(\vec{\alpha}).$$

where z is a number variable, and '$\simeq$' means that the l.h.s. is defined iff the r.h.s. is. In fact $\mathbf{Ind}(\mathcal{F})$ is equivalent to Kleene's S1–S9, since the $\mathbf{FP}$ schema can be obtained from the functional form of Kleene's recursion theorem.

SF ends this section by posing the question: Is there an interesting *relational* inductive theory over $\mathcal{N}$ – or rather over $\mathcal{S} = \langle S_n \rangle_n$, where $S_0 = \mathbb{N}$ and S_{n+1} is the power set of S_n?

In Section 4, SF briefly considers what he calls *axiomatic enumerative g.r.t.*, characterized by an axiomatic approach to Kleene's S9, and developed by Wagner and Strong [Str71], Moschovakis [Mos71], Fenstad [Fen74], Hyland [Hyl75] and others. It is shown in [KM77] that under quite general hypotheses a theory $\mathbf{Ind}(\mathcal{F})$ is an enumerative g.r.t. SF remarks on what he considers two defects of such enumerative approaches: (1) the *ad hoc* character of such codings, and (2) the necessity to incorporate $\mathcal{N}$ as part of the structure.

Section 5 gives some ideas ("which remain to be developed") for more restricted kinds of inductive schemata given by syntactic closure conditions or inference rules, where the course of the induction can be represented by a *derivation* or *computation* tree.

To take an example, consider *syntactic closure conditions*, and the *relational case*. We take a formula Γ in a language $\mathcal{L}$ over a structure $\mathcal{A} = \langle \langle A_i \rangle_{i \in I}, \ldots \rangle$ augmented by relation parameters which occur only positively in Γ. This gives a monotone schema Φ_Γ with a (least) fixed point, defining a function from tuples of relations (of the correct arity) to relations. We can then identify $\mathcal{F}$ with the class of such formulas Φ, and so $\mathbf{Ind}(\mathcal{F})$ is the set of functions so defined. SF gives two examples.

(1) Let $\mathcal{A} = (\mathbb{N}, 0, \boldsymbol{Sc})$, and let $\mathcal{F}$ be the class of *existential formulas* (positive in their relational parameters). Then $\mathbf{Ind}(\mathcal{F})$ is the class of all r.e. relations; cf. the Post-Smullyan approach [Smu61].
(2) Let $\mathcal{A}$ be any structure, $\mathcal{L}$ the corresponding full first order language, and $\mathcal{F}$ all formulas positive in their relation parameters. Then $\mathbf{Ind}(\mathcal{F})$ is the set of relations inductively definable over $\mathcal{A}$ in the sense of Moschovakis [Mos74].

As another example, we can associate, with certain inductive definitions, *rules of inference* and *derivation trees*. The inductive definition has the form: X is the least solution of $\Phi(X) \subseteq X$, i.e. the least set such that $\forall \vec{x}\big[\Gamma(\vec{x}, X) \to \vec{x} \in X\big]$, where elements that are put into X at any stage are related by elementary conditions (given by Γ) to elements already in X. We can write such a *closure condition* as an *inference rule* $\Gamma(\vec{x}, X)/X(\vec{x})$. Such inference rules give rise to a (possibly infinite) *derivation tree* for generating membership of X.

We turn to Part II (and Section 6), which deals with recursion on structures of *continuous functionals* over $\mathcal{N}$. SF sketches two similar constructions by Kleene [Kle59a] and Kreisel [Kre59] of a finite type structure $\mathcal{C} = \langle C_n \rangle_n$ of *hereditarily* (*total*) *continuous functionals* over $C_0 = \mathbb{N}$. (Another approach: the structure $\mathcal{C}^\smile$ of *hereditarily partial continuous* functionals, developed by Ershov, will be discussed below shortly.)

The principal question now is: are the resulting theories equivalent to those given by inductive schemata?

Considering first, *total continuous* (or "countable", to use Kleene's terminology) functionals: we consider functionals α^n of type n, where the value of α^{n+1} at any β^n is given by a finite amount of information U^n about β, given by "formal neighborhoods" which can be coded as natural numbers. The details are available in SF's paper, and, of course, in [Kle59a, Kre59]. Hence for all n, any such α^n can be *represented* by a type-one "associate" denoted by $\alpha^{(n+1)}$ or (here) by $\ulcorner \alpha \urcorner$.

The main difference between Kleene's and Kreisel's approach is that $\alpha^{n+1}(\beta^n)$ makes sense for all $\beta^n \in \mathbb{N}_n$ in Kleene's approach, but only for $\beta^n \in C_n$ in Kreisel's.

A functional $\varphi \in \mathcal{C}$ is said to be *recursively continuous* (or *recursively countable*) if it has a (total) recursive associate $\ulcorner \varphi \urcorner$. With Kleene's identification of C_n with part of $\mathbb{N}_n$, a central problem was to find the relationship between recursively continuous functionals and those generated by Kleene's schemata S1–S9.

In one direction: Kleene showed [Kle59a] that if φ is generated by his schemata, and *if φ is total on $\mathcal{C}$*, then φ (restricted to arguments in $\mathcal{C}$) is recursively continuous.

The question was raised by Kreisel [Kre59] if the converse holds, in the sense that every recursively continuous function on $\mathcal{C}$ is the restriction of a function generated by S1–S9. A counterexample was found by Tait (unpublished), namely a modulus of uniform continuity functional at type level 3.

The situation with K-K (Kleene-Kreisel) recursiveness is thus not satisfactory as in stands. In Section 7, SF points to a possible way forward, by turning to a more

general theory of higher-order *partial recursion*, which would reduce to the theory of K-K recursiveness in the special case of total functions.

The problem here is that even a *definition* for such a concept is problematic. He presents one proposed by Robert Winternitz, a former student of his at Stanford. (We omit details, but urge the reader to consult this paper.) This definition satisfies (among other good properties) an *enumeration* or *universality* property at all types.

Also important in the further development of the theory is the concept of *potential partial recursiveness*, which is satisfied by a function φ if there is some partial recursive continuous $\psi \supseteq \varphi$. With this definition, Kleene's partial positive result above can be can be re-cast in the form: Each φ which is (S1–S9) partial recursive on $\mathcal{N}$ has its restriction to $\mathcal{C}$ potentially partial recursive. (But see below.)

SF writes here: "I consider the main defect of this work on schemata to be taking $\mathcal{N}$ as the point of departure, rather than working entirely in the context of $\mathcal{C}$". This makes a difference in interpreting Kleene's schema S8:

$$\varphi(\alpha^{n+2}, \vec{\gamma}) \ \simeq \ \alpha(\lambda\beta^n \cdot \psi(\alpha, \beta, \vec{\gamma}))$$

in which even though α^{n+2} and $\vec{\gamma}$ range over $\mathcal{C}$, the "abstracted" variable β^n is taken to range over all of $\mathcal{N}$ (of the appropriate type). It would be more appropriate to have β^n ranging over $\mathcal{C}$. This would make the r.h.s. (and hence the l.h.s.) of this equation definable in more cases. (The other schemata, S1–S7 and S9, are not affected by such re-interpretation.) Then, as before, every functional generated by S1–S9 in this new interpretation on $\mathcal{C}$ is potentially partially recursive, and we can again inquire about the converse. However the uniform continuity functional at type level 3 still provides a counterexample.[6]

Section 8 deals with the other approach to computation on higher types noted above, based on the structure $\mathcal{C}^{\smile} = (C_n^{\smile})_n$ of hereditarily partial continuous functionals developed by Ershov [Ers72]. Here $C_0^{\smile} = \mathbb{N}$, and $C_{n+1}^{\smile}$ is the set of continuous partial functions from $C_n^{\smile}$ to $\mathbb{N}$ (suitably defined). Then C_n can be successively mapped into $C_n^{\smile}$ [Ers74]. To quote SF again: "Now there is also a natural definition of *partial recursive functional* on $C^{\smile}$. I studied the schematic generation of these functionals in [Fef77a], centering attention on so-called "search" operators introduced in Moschovakis [Mos69], namely $\nu x[\psi(x, \vec{\alpha}) \simeq 0]$ which is interpreted as 'an x such that $\psi(x, \vec{\alpha}) \simeq 0$'. By S10 we mean the scheme

$$\varphi(\vec{\alpha}) \ \simeq \ \nu x[\psi(x, \vec{\alpha}) \ \simeq \ 0]$$

which, without further restriction, must lead to multivalued functions ..." Single-valuedness can be recovered by a suitable restriction in the use of this scheme, resulting in a scheme (S10!).

The main results of [Fef77a] were that the multi-valued partial recursively continuous functionals over $\mathcal{C}^{\smile}$ are exactly those generated by (S1-S8) + (S10), and the single-valued ones are those generated by (S1-S8) + (S10!). However this is not completely satisfactory, since (S10!) is not a monotone scheme.

This result was improved by Winternitz, by incorporating the "strong or" operator introduced in [Pla66][7]:

$$\mathsf{OR}^+(\varphi_1, \varphi_2) \simeq 0 \iff \varphi_1(0) = 0 \ \vee \ \varphi_2(0) = 0. \tag{1.1}$$

[6] Gandy to SF, personal communication.

[7] The notation and presentation have been changed here to match that in [Fef96], discussed in §3 below.

Winternitz showed that

at type level 2, the functionals generated by (S1–S9)+OR^+ *are exactly the partial recursive functionals over* $\mathcal{C}^\smile$.

However the result does not hold at higher type levels. The solution turns out to come from work of Sazonov [Saz76] who introduced the functional $\exists^3$, defined by

$$\exists^3(\alpha^2) \simeq \begin{cases} 0 & \text{if } \alpha(\Lambda^1) \simeq 0 \\ 1 & \text{if } \alpha(\delta_n) \simeq 1 \text{ for some } n \end{cases}$$

where Λ^1 is the completely undefined function at level 1, and $\delta_n(x) \simeq z \iff x = 0 \wedge z = n$. Sazonov's result can then be formulated (in this framework) as:

A (partial) functional is partial recursively continuous over $\mathcal{C}^\smile$ *iff it is generated by the schemata* (S1–S9) + OR^+ + $\exists^3$.

This shows that *partial recursion over* $\mathcal{C}^\smile$ is equivalent to a *monotonic inductive schematic* theory.

Ershov's work is discussed again below in §3, in connection with SF's providing recursion-theoretic interpretations for abstract computational procedures over his stream algebras.

In Section 9 SF makes some concluding remarks:

> By [Ers74] the recursively continuous functions over $\mathcal{C}$ are restrictions to $\mathcal{C}$ of those over $\mathcal{C}^\smile$, and hence just those functionals total on $\mathcal{C}$ generated by the above schemata. However there remains the question of whether we can generate these functionals directly over $\mathcal{C}$. What we are really after are monotone schemata ...

Now it might at first be thought that a non-monotone theory could be found, for example (S1–S8) + (S10!). However, as Winternitz showed, although the partial recursively continuous functionals on $\mathcal{C}$ *are* closed under (S1–S7) and (S10!), they are *not* closed under (S8), even when abstracting over types 0 and 1 only. SF continues:

> With this we can complete our remark in Section 7 about Kleene's partial positive result ... [for] the statement of closure under (S1–S9) given in Section 7 above involves *potential* partial recursiveness in an essential way. ... The main question with which we are left is the following.

Question. Is there a natural monotone collection $\mathcal{F}$ over $\mathcal{C}$ such that the partial recursively continuous functionals over $\mathcal{C}$ are exactly those generated by $\mathbf{Ind}(\mathcal{F})$?

SF concludes this fascinating paper with the remark:

> As long as this question remains unsettled, the inductive schematic approach to g.r.t. is not completely vindicated. But I hope that the considerations in this paper combined with the detailed work of [KM77] and [Mos76], which demonstrate its scope otherwise, will lead one to give serious attention to this program.

2. A new approach to abstract data types, parts I and II

In this pair of papers [Fef92a, Fef92b], SF switches gears, and focuses on computation not on the classical structures (such as naturals or ordered reals, and higher types on these) but on *abstract data types*, which can be taken (for now) as classes of algebras of a given signature Σ, closed under Σ-isomorphism. SF motivates the topic by saying: "The concept of abstract data types (ADTs) has emerged in the

last fifteen years or so as one of the major programming design tools, with the emphasis on modular construction of large-scale programs."

The first paper (Part I) is an informal introduction to ADTs, while the second gives a more formal development, with the emphasis again on computation over these. We thus give a very brief overview of Part I, as an introduction to Part II, which is our main concern.

SF begins Part I by stating:

> [J]ust as in mathematics generally, one is concerned in computational practice with general algebraic notions such as orderings, rings, fields, polynomials, etc. There is additional concern in computer science with other kinds of ADTs such as lists, stacks, trees, records, arrays, streams, etc. A coherent account of how these are all to be treated for computational purposes requires answers to such questions as:
>
> Q1. What are ADTs and how may they be specified?
> Q2. What does it mean to implement an ADT?
> Q3. How can we construct new ADTs from old ones?
> Q4. What does it mean to compute with ADTs?

SF's aim is to answer the above four questions, especially Q4, in the course of this and the following paper. Since this paper gives only a semi-formal development, we discuss it very briefly, before turning to Part II for a more detailed account.

SF refers to previous foundational approaches: algebraic [GTWW77], computational or recursion-theoretic [BT83, TZ88] and type-theoretic [MP85]. He argues that all previous foundational approaches fail in certain cases.

> A new, constructively based, approach is proposed here to provide a sufficiently general account ... within the conceptual framework of the school of constructive mathematics associated with Bishop ... [Bis67]. The formal foundations will be provided by theories of operations and classes in which that style of constructive mathematics can be formalized ... [T]hese will be taken up in Part II.

SF gives some examples of ADTs, noting that "there is a basic division into those structures whose objects are described or generated in a finitary way, and those whose description is infinitary." Examples of finitary ADTs dicussed are: lists over arbitrary types; lists over preordered types; binary and finitely branching trees; and finite sets. Examples of infinitary ADTs are *infinite streams* and *infinite precision reals* – the latter following Bishop's implementation [BB85, pp. 18–19].

We turn to Part II. As SF says in the opening sentence, the main purpose of this paper is to give a precise definition of *abstract computation procedures*[8] F on ADTs, with interpretations $\mathsf{F}^{\mathcal{A}}$ over structures

$$\mathcal{A} \;=\; (A_0, \ldots, A_n,\; =_{A_0},\; \ldots,\; =_{A_n},\; F_0, \ldots, F_m) \tag{2.1}$$

of signature Σ, say, where $=_{A_i}$ is an "equality" relation[9] on A_i $(i = 0, \ldots, n)$, A_0 is the boolean type $\mathbb{B} = \{\mathtt{tt}, \mathtt{ff}\}$, $=_{A_0}$ is the identity on $\mathbb{B}$, and F_j is either a constant (0-ary function), partial (level 1) function or partial (level 2) functional over $\mathcal{A}$ of a specified arity. The constants of $\mathcal{A}$ include $\mathtt{tt}$ and $\mathtt{ff}$, and the functions and functionals among F_j *preserve equality* and are *monotonic*.

The aim here is to clarify the concept of *abstract computational procedure* F satisfying, for any $\mathcal{A}$ as above, the following criteria:

[8] Written 'π' in [Fef92a]. Notation changed here to match [Fef96]; cf. §3 below.

[9] I.e. an equivalence relation w.r.t. the F_k's.

C1. F associates with $\mathcal{A}$ an object $\mathsf{F}^{\mathcal{A}}$ of specified arity over $\mathcal{A}$.
C2. $\mathsf{F}^{\mathcal{A}}$ is determined by the (individual, function and functional) constants of $\mathcal{A}$.
C3. The map $\mathcal{A} \mapsto \mathsf{F}^{\mathcal{A}}$ preserves Σ-isomorphism.
C4. $\mathsf{F}^{\mathcal{A}}$ preserves the equality relations on $\mathcal{A}$.
C5. For domains A_i of $\mathcal{A}$ contained in $\mathbb{N}$, $\mathsf{F}^{\mathcal{A}}$ reduces to an ordinary computational procedure (see Remark 2 below).

SF explains:

> These requirements are met here by a *generalized recursion theory*[10] (g.r.t.) which provides a notion of computability over arbitrary structures of the kind described above. In order to satisfy C3 we must insure that whenever an object is defined by recursion it is uniquely specified. For (partial) functions this will be as a least fixed point (LFP) of a suitable monotonic functional. There are two forms of g.r.t. available in the literature which feature LFP as a central scheme ... namely those of Moschovakis [Mos84, Mos89] and the earlier Platek [Pla66].[11]

In both of these definition by recursion is implemented as the least fixed point of a suitable monotonic functional. SF adapts Moschovakis's version, but with schemata like Kleene's S1–S9 [Kle59b], with S9 replaced by a simple LFP recursion.

SF continues:

> The g.r.t. developed here ... applies to a wide variety of data universes V, with weak closure conditions on the classes of partial functions and functionals over V. There are two extremes of interpretation: (i) V is the full cumulative hierarchy and "all" functions and functionals are admitted; (ii) $V = \omega$ [12], and we only admit partial recursive functions and functionals. The setting (i) is the usual one for g.r.t., while the setting (ii) serves for the precise formulation of the criterion C5; the statement of that is the main new conribution of this paper.

Remarks.

(1) We need only consider structures as in (2.1) with function(al)s up to type level 2, because of Platek's result given in §1 above.
(2) By "ordinary computation procedure", SF means the structure constructed by Kleene over $\mathbb{N}$ in [Kle59b]. Also recall footnote 4 for the concept of partial recursive functional.

A structure $\mathcal{A}$ has a "data universe": an underlying universe $\boldsymbol{V}$ containing the data objects $a, b, \ldots, x, y, \ldots$, a collection $\boldsymbol{DT}$ of subsets $A, B, \ldots, X, Y, \ldots$ of $\boldsymbol{V}$, called *data types*, containing $\mathbb{B}$, $\mathbb{N}$ and $\boldsymbol{V}$, and closed under Cartesian product.

There is a collection $\boldsymbol{PFn}$ of *partial functions* $\varphi, \psi, \ldots$ on $\boldsymbol{V}$, and more specifically, collections of partial functions[13] $\varphi\colon A \stackrel{\sim}{\to} B$ between the various data types.

$\mathcal{A}$ has (as in (2.1) above) *basic domains* $A_0, \ldots, A_n$, with $A_0 = \mathbb{B}$. We let i, j, k range over $\{0, \ldots, n\}$ and $\bar{\imath}, \ldots,$ over finite (possibly empty) sequences of these. For $\bar{\imath} = (i_1, \ldots, i_\nu)$, put $A_{\bar{\imath}} = A_{i_1} \times \cdots \times A_{i_\nu}$. Then a partial function φ on $\mathcal{A}$ has *arity* $\bar{\imath} \to j$ if $\varphi\colon A_{\bar{\imath}} \stackrel{\sim}{\to} A_j$. We let $\sigma, \tau, \ldots$ range over arities of partial functions

[10] Emphasis added. The meaning of this phrase is discussed below.
[11] As noted in §1 above.
[12] Below I use '$\mathbb{N}$' for 'ω'.
[13] This notation '$\stackrel{\sim}{\to}$' for partial functions is not the same as SF's here, but is used for consistency with his notation in the paper [Fef96] discussed in §3 below.

on $\mathcal{A}$. For $\bar{\sigma} = \sigma_1, \ldots, \sigma_\mu$, put $A_{\bar{\sigma}} = A_{\sigma_1} \times \cdots \times A_{\sigma_\mu}$. Then a partial functional F on $\mathcal{A}$ has *arity* $(\bar{\sigma}, \bar{\imath}) \to j$ if $F\colon A_{\bar{\sigma}} \times A_{\bar{\imath}} \xrightarrow{\sim} A_j$. When $\varphi = (\varphi_1, \ldots, \varphi_\mu)$ and $x = (x_1, \ldots, x_\nu)$, we write $F(\varphi, x)$ for $F(\varphi_1, \ldots, \varphi_\mu, x_1, \ldots, x_\nu)$. Generally F has type level 2, but when $\mu = 0, \nu > 0$ then F is a partial function on $\mathcal{A}$ of level 1, and when $\mu = \nu = 0$ then F is a constant of sort j of level 0.

Suppose $\mathcal{A}$ (as in (2.1) above) has the basic functionals

$$F_k\colon A_{\bar{\sigma}_k} \times A_{\bar{\imath}_k} \to A_{j_k} \ (k = 1, \ldots, n).$$

Then $\mathcal{A}$ has the *signature* $\Sigma(\mathcal{A}) = \big(n,\ \langle \bar{\sigma}_k, \bar{\imath}_k, j_k\rangle_{1 \le k \le m}\big)$.

In Section 4, four interpretations of the structure are presented, which we describe here, (mainly in SF's own words). For convenience I refer to these as four "models".

Model 1: The full set-theoretic interpretation.
"Here V is the class of all sets in the cumulative hierarchy. The types range over all sets in V ... Functionals are just those partial functions in V of the form $F(\varphi, x)$ where the φ_k's are partial functions in V."

Model 2: The set-theoretic interpretation on computational data.
"For computational purposes, all data should be represented in in finite symbolic form; without loss of generality, we can take the universe to be $V = \mathbb{N}$. ... The partial functions here are arbitrary $\varphi\colon \mathbb{N} \xrightarrow{\sim} \mathbb{N}$ ... Partial functionals $F(\varphi, x)$ in this interpretation take *arbitrary partial function arguments*[14] φ on $\mathbb{N}$. Special interest attaches below to those F which are *partial recursive* (p.r.) or have a p.r. extension. Note that p.r. functionals are *not closed under abstraction* when the remaining function arguments are not p.r."

Model 3: The recursion-theoretic interpretation, extensional form.
"Here again we take $V = \mathbb{N}$ .... The types ... range over arbitrary subsets of $\mathbb{N}$ (or, more generally, over any collection of subsets closed under arithmetical definability). The collection ***PFn*** is taken to be the p.r. functions on $\mathbb{N}$, and ***PFnl*** the p.r. functionals of *p.r. function arguments*[15]. Thus we have *closure under abstraction* in this case."

Model 4: The recursion-theoretic interpretation, intensional form.
This is like model 3, except that ***PFn*** consists of *Gödel numbers* or indices e of p.r. functions, with application given by $e\,x = \{e\}x$. Now ***PFnl*** coincides with ***PFn***, but with the emphasis on indices which are "extensional" or "effective" in the recursion-theoretic sense.

Models (2)–(4) are the main ones considered in this paper.

Returning to a general structure $\mathcal{A}$ (as in (2.1)), SF defines a *partial order* on ***PFn***: for φ, ψ of arity $\bar{\imath} \to j$ on $\mathcal{A}$:

$$\varphi \subseteq_A \psi \iff \forall x \in A_{\bar{\imath}} \big[\, \varphi(x) \downarrow \implies \psi(x) \downarrow = \varphi(x)\big]. \tag{2.2}$$

A partial functional F of arity $(\bar{\sigma}, \bar{\imath}) \to j$ is $\mathcal{A}$-*monotonic* if

$$\forall \varphi, \psi \in A_{\bar{\sigma}}, \forall x \in A_{\bar{\imath}} \big[\, F(\varphi, x) \downarrow \wedge\ \varphi \subseteq_A \psi \implies F(\psi, x) \downarrow = F(\varphi, x). \tag{2.3}$$

Next, for partial functionals F_1, F_2 of arity $(\bar{\sigma}, \bar{\imath}) \to j$ on $\mathcal{A}$, we define

$$F_1 \subseteq_A F_2 \iff \forall \varphi \in A_{\bar{\sigma}} \forall x \in A_{\bar{\imath}} \big[F_1(\varphi, x) \downarrow \implies F_2(\varphi, x) \downarrow = F_1(\varphi, x). \tag{2.4}$$

[14,15]Emphasis added. This gives the essential difference between models 2 and 3.

Next, in order to determine the *least fixed point* of a functional $G\colon A_\sigma \times A_{\bar\iota} \to \mathcal{A}_j$, where $\sigma = (\bar\iota \to j)$, we define $\widehat{G}\colon A_\sigma \to A_\sigma$ by $(\widehat{G}\varphi)x = G(\varphi, x)$ (i.e. a "curried" version of G). Then, supposing G is $\mathcal{A}$-monotonic, so is $\widehat{G}$, and further, a (unique) *least fixed point* $\mathsf{L}G$ of G can be found from $\widehat{G}$, as follows.

First, L is called an LFP operator on $\mathcal{A}$ if for any $\sigma = \bar\iota \to j$ and $\mathcal{A}$-monotonic $G\colon A_\sigma \times A_{\bar\iota} \overset{\sim}{\to} A_j$, we have:

(i) $\mathsf{L}G \in A_\sigma$ and $\widehat{G}(\mathsf{L}G) \subseteq_A \mathsf{L}G$;
(ii) whenever $\psi \in A_\sigma$ and $\widehat{G}(\psi) \subseteq_A \psi$ then $\mathsf{L}G \subseteq_A \psi$;
(iii) For $\mathcal{A}$-monotonic G_1, G_2 of the same arity, $G_1 \subseteq_A G_2 \implies \mathsf{L}(G_1) \subseteq_A \mathsf{L}(G_2)$.

The question of the *existence* of LFP operators in the various interpretations must still be discussed (see below).

Recall (2.1) we are assuming each domain A_i has an "equality" relation. Next SF defines the relation of $\mathcal{A}$-*equality* between partial functions:

$$\varphi =_{\mathcal{A}} \psi \iff \varphi \subseteq_A \psi \ \wedge\ \psi \subseteq_A \varphi$$

and hence the concept of a functional F *preserving* $\mathcal{A}$-*equalities*. Then F is said to be *strongly* $\mathcal{A}$-*monotonic* if it is $\mathcal{A}$-monotonic and also preserves $\mathcal{A}$-equalities.

We come to ***schemata*** for ***abstract computation procedures*** (***ACPs***) over Σ. Suppose given a Σ-structure $\mathcal{A}$, with basic functionals $F_1, \ldots, F_m$, where each F_k has a specified arity $\bar\sigma_k \times \bar\iota_k \to j_k$. We make some *general assumptions*:

(i) $A_0 = \mathbb{B} = \{\mathtt{tt}, \mathtt{ff}\}$, where $=_{A_0}$ is the identity relation, and
(ii) $\mathtt{tt}$ and $\mathtt{ff}$ are themselves in Σ.

A formal language of computational procedures over Σ is defined, with variables $a, \ldots, x, \ldots$ of all Σ-sorts, partial function varables $\varphi, \psi, \ldots$ of each arity, and functional symbols $\mathsf{F}, \mathsf{G}, \mathsf{H}, \ldots$ of each appropriate arity. It is further assumed:

(iii) For each $\mathcal{A}$, $\mathsf{L}^{\mathcal{A}}$ is an operator from $\mathcal{A}$-functionals of arity $\sigma \times \bar\iota \to j$, where $\sigma = \bar\iota \to j$, to A_σ-partial functions (for each $\bar\iota, j$).

Note that at this stage $\mathsf{L}^{\mathcal{A}}$ is not yet assumed to be an LFP operator on $\mathcal{A}$.

There follows a list of *schemata* (for Σ):

I	(Initial fns)	$\mathsf{F}(\varphi, x) \simeq \mathsf{F}_k(\varphi, x) \quad (k = 0 \ldots, m)$
II	(Identity)	$\mathsf{F}(x) = x$
III	(Application)	$\mathsf{F}(\theta, x) \simeq \theta(x)$
IV	(Conditional)	$\mathsf{F}(\varphi, x, v) \simeq [\mathsf{if}\ v = \mathtt{tt}\ \mathsf{then}\ \mathsf{G}(\varphi, x)\ \mathsf{else}\ \mathsf{H}(\varphi, x)]$
V	(Structural)	$\mathsf{F}(\varphi, x) \simeq \mathsf{G}(\varphi_f, x_g)$
VI	(Indiv. subst.)	$\mathsf{F}(\varphi, x) \simeq \mathsf{G}\big(\varphi, x, \mathsf{H}(\varphi, x)\big)$
VII	(Func. subst.)	$\mathsf{F}(\varphi, x) \simeq \mathsf{G}\big(\varphi, \lambda u.\mathsf{H}(\varphi, x, u), x\big)$
VIII	(LFP)	$\mathsf{F}(\varphi, x, u) \simeq \big[\mathsf{L}(\lambda\theta, w.G(\varphi, \theta, x, w))\big](u)$

In the above schemata, φ and x are (respectively) tuples of function and individual variables; in schema IV v is boolean variable; in schema V, f and g are (respectively) mappings of the indices of the variable tuples φ and x, and φ_f and x_g are the corresponding mappings of these tuples.

An *ACP* for Σ is then a partial functional F generated by the above schemata.

Note the resemblance of the above schemata (other than VIII) to Kleene's schemata [Kle59b] apart from S5 (primitive recursion on the integers) and S9 (enumeration).

So for each Σ-structure $\mathcal{A}$ and each ACP F generated by the schemata, there is an associated partial functional $\mathsf{F}^{\mathcal{A}}$, where the intended semantics is clear. Here we must make one more general assumption:

(iv) $\mathbf{L}^{\mathcal{A}}$ is an LFP operator on $\mathcal{A}$.

SF then states and proves his two main theorems:

Theorem 1 (Preservation of strong monotonicity).
If each F_k $(k = 1, \ldots, m)$ is strongly monotonic, then so is $\mathsf{F}^{\mathcal{A}}$ for every ACP F generated by the schemata I–VIII.

The proof amounts to showing that the property of strong monotonicity is preserved by an application of any of the schemata, including notably the LFP schema VIII.

Theorem 2 (Invariance under isomorphism).
For any Σ-ACP F, an isomorphism between two Σ-structures $\mathcal{A}$ and $\mathcal{A}'$ induces an isomorphism between $\mathsf{F}^{\mathcal{A}}$ and $\mathsf{F}^{\mathcal{A}'}$.

Note that Theorems 1 and 2 imply (respectively) criteria C4 and C3 of the five criteria listed above.

Next (in Section 10) we suppose given an $\mathcal{A}$-monotonic functional $G\colon A_\sigma \times A_{\bar{\iota}} \xrightarrow{\sim} A_j$, with $\sigma = (\bar{\iota} \to j)$. The construction of the LFP operator $\mathbf{L}^{\mathcal{A}}(G)$ is shown for each of the four types of interpretation $\mathcal{A}$ considered in Section 4 (see above).

Briefly: in the case of all four models, $\mathbf{L}^{\mathcal{A}}(G)$ is constructed as the union of a transfinite sequence of approximations from below. In the case of model 4, the Myhill-Shepherdson Theorem [MS55] is needed.

We come now to the final important result of the paper (in Section 11), which shows how computation on ADTs can reduce (under certain conditions) to ordinary computation.

SF restricts consideration to models with $V = \mathbb{N}$, and the extensional version of the recursion-theoretic interpretation (model 3). So we assume that all the domains of $\mathcal{A}$ are contained in $\mathbb{N}$, and all the F_k are strongly $\mathcal{A}$-monotonic. SF then shows (**Theorem 3**) that for each ACP F, $\mathsf{F}^{\mathcal{A}}$ can be taken as the p.r. functional $\mathsf{F}^{\mathcal{V}}$ on $\mathcal{A}$. Hence $\mathsf{F}^{\mathcal{A}}$ satisfies criterion C5, and thus all the desired criteria C1–C5 for ACPs.

Another version of this theorem will be encountered as Theorem 6 in [Fef96], discussed in §3 below.

In a concluding section, SF presents some applications of this theory, revisiting some of his examples in Part 1. As he says:

> [A]ll the standard finitary examples of ADTs, such as lists, trees, sets, etc., as described in Part I of this paper ... have implementations whose F_k are simply partial recursive functions.

"Infinitary" data types, such as infinite streams and infinite precision reals, are discussed briefly. In the case of infinite streams over a structure $(A, =_A)$, SF uses the following structure, first defined in Part 1 above:[16]

$$\boldsymbol{Stream}(A) \;=\; (S,\, A,\, \mathbb{N},\, =_S,\, =_A,\, =_{\mathbb{N}},\, \ldots,\, \boldsymbol{Cons},\, \boldsymbol{Hd},\, \boldsymbol{T\ell},\, \boldsymbol{Sim}) \tag{2.5}$$

This has three domains, A, S and $\mathbb{N}$ for data, streams of data and naturals respectively; the standard operations on $\mathbb{N}$, shown here as '...', and the stream operations

[16] The notation here has been modified to conform to that in [Fef96], cf. §3 below.

$\boldsymbol{Cons}\colon A \times S \to S$, $\boldsymbol{Hd}\colon S \to A$ and $\boldsymbol{Tl}\colon S \to S$, where (informally)

$$\begin{aligned} \boldsymbol{Cons}\,(b,\, \langle a_0, a_1, a_2 \ldots\rangle) &= \langle b, a_0, a_1, a_2, \ldots\rangle \\ \boldsymbol{Hd}\,\langle a_0, a_1, a_2, \ldots\rangle &= a_0, \\ \boldsymbol{Tl}\,\langle a_0, a_1, a_2, \ldots\rangle &= \langle a_1, a_2, \ldots\rangle. \end{aligned}$$

In order to characterize infinite stream structures up to isomorphism, we also need a second order "simulating" functional $\boldsymbol{Sim}\colon (\mathbb{N} \to A) \to S$ which is a bijection from the set of "all" functions $f\colon \mathbb{N} \to A$ onto S. (Without this, we can only ensure the existence of eventually constant streams.)

SF comments: "Here, it seems, only the intensional recursion-theoretic interpretation[17] is appropriate." This is accomplished by interpreting S as the set of all indices of total recursive functions from $\mathbb{N}$ to $\mathbb{N}$, and $\boldsymbol{Sim}$ as the identity on $\mathbb{N}$.

The ADT of infinite precision reals is likewise given an intensional recursion-theoretic interpretation, in which the reals are interpreted as indices of effective Cauchy sequences.

This perspective will shift quite dramatically in the following paper [Fef96], with the investigation of higher order ("extensional") models of streams, and a second order '$\boldsymbol{Sim}$' operator.

3. Computation on ADTs: the extensional approach

This paper carries the dedication "With profound gratitude to Stephen C. Kleene" (then recently deceased) with the footnote "Kleene was not my mentor, official or otherwise, but through his exceptional development of our subject I learned as much from him as if he had been."

I quote from the introduction:

> This paper is a continuation of the work of Feferman [Fef92a, Fef92b] which initiated an approach through a form of *generalized recursion theory* (g.r.t.) to computation on *abstract data types* (ADTs), including intensionally presented types ...
>
> [W]e separate out the extensional part of the theory and show how it may be applied to computation on streams as an ADT. One of the main new contributions here is an explanation of how this is to be done for finite "nonterminating" streams as well as infinite streams, and even more general partial ("gappy") streams.

Logically (as stated in the previous section) an ADT is just a class of structures closed under isomorphism. "[O]ne is mainly interested in structures determined by categorical or relatively categorical conditions." Paradigmatic examples considered are, for any data set A: A-lists and A-streams.

Continuing with the framework of the previous paper (§2 above). the general theory in this paper applies, for a given signature Σ, to many-sorted Σ-structures

$$\mathcal{A} \;=\; (A_0, \ldots, A_n,\; F_1, \ldots, F_m) \tag{3.1}$$

where $A_0 = \mathbb{B} = \{\mathtt{tt}, \mathtt{ff}\}$, and each F_j is a partial functional (or function, or constant, including $\mathtt{tt}$ and $\mathtt{ff}$) of type level ≤ 2. It will also be assumed that each F_j is $\mathcal{A}$-*monotonic*.

[17] That is, model 4 described above.

As SF says, "computation on streams is subsumed under a general theory of computation for arbitrary structures[18]."

Remark (Treatment of equality). Note the difference between the structures displayed here (3.1) and in (2.1): here an ("intensional") equality predicate is *not* automatically assumed at each sort. In fact, infinite stream equality is generally non-computable. SF turns to this issue later in the paper.

In Section 3 a system of formal schemata for a given signature Σ is presented. This is the same as the system of schemata (I–VIII) given in [Fef92b] and shown above (§2); and, as before, with each Σ-structure $\mathcal{A}$ and each ACP F generated by the schemata, is associated a partial functional (or function, or constant) $\mathsf{F}^{\mathcal{A}}$ of type level ≤ 2.

As shown in [Fef92b] (cf. §2 above), an $\mathcal{A}$-monotonic functional has an associated *least fixed point* (LFP), which is also monotonic, by the **LFP Monotonicity Lemma**[19]. Hence for schema VIII to make sense on $\mathcal{A}$, it must be verified that $\mathsf{F}^{\mathcal{A}}$ is $\mathcal{A}$-monotonic. More generally, it is shown (**Monotonicity Lemma**[20]) that the ACPs generated by all the schemata are $\mathcal{A}$-monotonic, assuming that the initial functionals $F_1, \ldots, F_m$ are.

Remarks (Extensionality). The assumption of extensionality in this paper leads to two interesting divergences from the theory developed in the previous paper [Fef92b] (§2), with intensional equality:

(1) In [Fef92b] the significant property of ACPs was *strong monotonicity*, i.e., monotonicity plus equality-preservation. Here it is simply monotonicity.
(2) In [Fef92b] the existence of the LFP operator had to be explicitly assumed ("General assumption (iv)" at the beginning of Section 8) and proved for the four types of interpretations ("Models 1–4"). Here the construction of the LFP operator (the LFP Monotonicity Lemma) can be shown quite generally as a consequence of the more general Monotonicity Lemma[21]. Again, it is constructed as the union of a transfinite sequence of approximations from below.

Back to Section 3: It is next shown that the schemata are *invariant under isomorphism*, i.e., if $\mathcal{A}$ and $\mathcal{A}'$ are Σ-isomorphic, then so are $\mathsf{F}^{\mathcal{A}}$ and $\mathsf{F}^{\mathcal{A}'}$ for all F generated by the schemata. This justifies the terminology "*abstract* computation procedures", and the notation $\mathbf{ACP}(\Sigma)$ for the collection of ACPs generated by the schemata over Σ, and $\mathbf{ACP}(\mathcal{A})$ for the collection of all $\mathsf{F}^{\mathcal{A}}$ for any Σ-structure $\mathcal{A}$.

Next (Section 4) this paper deals with an important *substructure property* of Σ-structures.

First, some definitions. Suppose given a Σ-structure $\mathcal{A}$ as in (3.1) above, and subsets $B_i \subseteq A_i$ for $i = 1, \ldots, n$. Write $A = (A_0, \ldots, A_n)$ and $B = (B_0, \ldots, B_n)$.

For a level-1 partial function φ over A, the *restriction* $\varphi \restriction B$ and the concept "B is *closed under* φ" are defined in the standard way.

---

[18] as in (3.1).
[19,20] My terminology.
[21] Or more accurately, simultaneously with this lemma.

For a level-2 partial functional $F\colon A_{\bar\sigma} \times A_{\bar\imath} \to A_j$, the *restriction* $F \restriction B$ means the function $\lambda\varphi \in B_{\bar\sigma} \cdot \lambda x \in B_{\bar\imath} \cdot F(\varphi, x)$, and B is said to be *closed under* F if[22]

$$\forall \varphi \in A_{\bar\sigma}, \forall x \in B_{\bar\imath}\, \big[B \text{ closed under } \varphi \ \wedge\ F(\varphi, x)\downarrow \implies F(\varphi, x) \in B_j \ \wedge\ F(\varphi \restriction B,\, x) = F(\varphi, x)\big].$$

We then say that B *determines a substructure of* A if $B_0 = A_0 = \mathbb{B}$ and B is closed under F_k for $k = 1, \ldots, m$.

SF then states and proves his

Substructure Theorem (Theorem 1 + Corollary).
Suppose B determines a substructure of $\mathcal{A}$. Then putting

$$\mathcal{B} \ =\ (B_0, \ldots, B_n,\ F_1 \restriction B, \ldots, F_m \restriction B),$$

B is closed under $\mathsf{F}^{\mathcal{A}}$ for each Σ-ACP F, and

$$\mathsf{F}^{\mathcal{B}} \ =\ \mathsf{F}^{\mathcal{A}} \restriction B.$$

This result turns out to be very useful for the rest of the paper, as we will see.

Next (Section 5) the paper deals with *continuity* of functionals. A functional F on $A = (A_0, \ldots, A_n)$ of type level 2 is said to be *continuous* if for any φ, x, y,

$$F(\varphi, x) \simeq y \implies \exists \text{ finite } \tilde\varphi \subseteq \phi : F(\tilde\varphi) \simeq y.$$

SF next states and proves the

Continuity Theorem (Theorem 2). If each basic functional F_k of $\mathcal{A}$ is continuous, then for each ACP F, $\mathsf{F}^{\mathcal{A}}$ is continuous.

The proofs of the Substructure and Continuity Theorems are (as one would expect) by induction on the generation of F by the schemata. However they are far from routine.

Next there is a small section (Section 6) on computation on *first-order structures*.

A signature $\Sigma = \big(n,\ \langle \bar\sigma_k, \bar\imath_k, j_k\rangle_{1\le k\le m}\big)$ is said to be *first-order* if $\bar\sigma_k$ is empty for $k = 1, \ldots, m$. In that case Σ-structures $\mathcal{A} = (A_0, \ldots, A_n,\ F_1, \ldots, F_m)$ are *first-order* in the sense that each F_k has type level 1 or 0.

In that case also, all the F_k are vacuously *monotonic* and *continuous*, and so (by the Monotonicity Lemma and Continuity Theorem) all ACPs over $\mathcal{A}$ are monotonic and continuous.

Further, for computation on *first-order* structures, schema VII (for function substitution) can be omitted, since (as SF shows) if Σ is a first-order signature, then the system $\mathbf{ACP}_0(\Sigma)$ of ACPs over Σ obtained by omitting schema VII is closed under that schema.

As an example, consider the "*ur*-structure for recursion theory"

$$\mathcal{N} \ =\ (\mathbb{N},\ \boldsymbol{Sc},\ \boldsymbol{Pd},\ \mathbf{0},\ \boldsymbol{Eq_0}). \tag{3.2}$$

where the booleans are coded as $\{0, 1\}$, $\boldsymbol{Pd}$ is the predecessor function with $\boldsymbol{Pd}(0) = 0$, and $\boldsymbol{Eq_0}$ tests for equality with 0.

SF then states and proves the following interesting theorem. (For the concept of "partial recursive (p.r.) functional", see footnote 4.)

Theorem 5 (Characterizing ACP($\mathcal{N}$)).

[22] As SF points out, this implies that $F \restriction B\colon B_{\bar\sigma} \times B_{\bar\imath} \xrightarrow{\sim} B_j$, but not conversely.

(i) The ACPs of type level 1 over $\mathcal{N}$ are exactly the p.r. functions.
(ii) The ACPs of type level 2 over $\mathcal{N}$ are exactly the p.r. functionals when restricted to total function arguments.
(iii) Every ACP of type level 2 over $\mathcal{N}$ is p.r. on partial function arguments, but not conversely.

A counterexample for the failure of the converse for (iii) is given by the "strong or" functional OR^+ (cf. (1.1)) as shown by Platek [Pla66]. In fact, the equivalence

$$\mathbf{ACP}(\mathcal{N}) + \mathsf{OR}^+ \iff \text{p.r. on } \boldsymbol{PFn}(\mathcal{N})$$

for level 2 functionals was proved by Winternitz.[23]

Next, SF defines the concept of ***partial recursive structure in*** $\mathbb{N}$. This is a structure of the form

(i) $\mathcal{A} = (A_0, \ldots, A_n,\ F_1, \ldots, F_m)$, where
(ii) each $A_i \subseteq \mathbb{N},\ A_0 = \mathbb{B} = \{0, 1\}$.
(iii) each F_k is the restriction to $A = (A_0, \ldots, A_n)$ of a p.r. functional F_k^* on $\mathbb{N}$, under which A is closed.

Using the Substructure Theorem, SF then derives:

Theorem 6 (ACPs of p.r. structures in $\mathbb{N}$). Suppose $\mathcal{A}$ is a p.r. structure in $\mathbb{N}$ of signature Σ. Then for each F in $\mathbf{ACP}(\Sigma)$, $\mathsf{F}^{\mathcal{A}}$ is the restriction to $\mathcal{A}$ of a p.r. functional F^*, and is (therefore) continuous.

This provides another version of Theorem 3 in [Fef92b, Sec. 11] (cf. §2 above). To convey the significance of this result (and of the Substructure Theorem), let me quote SF here:

> Most examples of abstract data types ... which contain partial recursive structures are those whose domains are generated by finitely many finitary operations, or are obtained from such by restriction, such as lists, finites sets, finite trees, records, etc. ...
> [I]f $A = \{a_0, a_1, \ldots, a_n, \ldots\}$ is any countable set, we can realize lists-of-A's as a partial recursive structure, no matter how A is identified as a subset of $\mathbb{N}$ ... For example, ... A might be a nonrecursive subset of $\mathbb{N}$, such as the set of Gödel numbers of total recursive functions ... That is why no restriction was made on the A_i's in the definition [of p.r. structures in $\mathbb{N}$] other than that they be subsets of $\mathbb{N}$.

The next section (8) illustrates the theory of Section 7 with the ADT of lists over a structure A. To quote SF: "The case of abstract computational procedures on (relativized) list structures is paradigmatic for finitary data types in many respects, and is useful for comparison with computation on infintary data types, of which streams form the main example in this paper."

SF shows how all the standard list operations can be defined as ACPs. He presents a number of formulations of definition by *list recursion*, and demonstrates the use of the Substructure Theorem in the case of p.r. list structures.

We will not describe this development in detail, moving rather on to the next section (9) dealing with the infinitary data type of *streams*.

This (together with the following sections) is the most interesting part of the paper. It deals with A-streams, or (potentially) infinite sequences of members of A.

[23] This has already been discussed in [Fef77b] (cf. §1 above).

Keeping to the framework of computation on ADTs, SF develops and investigates the structure $\mathcal{S}$ of A-streams with basic sets A and S.

To quote SF: "[T]hough a standard interpretation of S consists of *second-order* objects, in the present approach they are to be treated as *first-order* objects in $\mathcal{S}$."

However, treating streams as first-order objects, like lists, leads to trouble. For consider an axiomatization of a first-order structure of streams:[24]

$$\mathcal{S}^{(1)} \;=\; (A,\, S,\; \boldsymbol{Cons},\, \boldsymbol{Hd},\; \boldsymbol{T\ell}) \tag{3.3}$$

(the superscript "1" indicating a *first-order* structure) where

(i) $A \neq \emptyset$

(ii) $\boldsymbol{Cons}\colon A \times S \to S, \quad \boldsymbol{Hd}\colon S \to A, \quad \boldsymbol{T\ell}\colon S \to S,$

(iii) $\forall a \in A \forall s \in S\big[\boldsymbol{Hd}(\boldsymbol{Cons}(a,s)) = a \wedge \boldsymbol{T\ell}(\boldsymbol{Cons}(a.s)) = s\big].$

As SF writes: "The main point against this is that these (and similar) conditions do not uniquely determine $\mathcal{S}^{(1)}$ up to isomorphism, given A. Two nonisomorphic structures are obtained by interpreting S in the first instance to be the set $(\mathbb{N} \to A)$ of *all* functions from $\mathbb{N}$ to A, and in the second instance to be the subset $(\mathbb{N} \xrightarrow{\mathsf{fin}} A)$ ... of eventually constant functions."

These two structures will be denoted here, respectively, by $\mathcal{S}^{(1)}[\mathbb{N} \to A]$ and $\mathcal{S}^{(1)}[\mathbb{N} \xrightarrow{\mathsf{fin}} A]$, the latter clearly a substructure of the former.

The second problem with this approach is that these conditions do not guarantee closure under some standard computation procedures, such as recursion on streams. One can easily find examples of recursively defined functions from $\mathbb{N}$ to A which are are not in $\mathcal{S}^{(1)}[\mathbb{N} \xrightarrow{\mathsf{fin}} A]$, and hence (by the Substructure Theorem) also not ACPs in $\mathcal{S}^{(1)}[\mathbb{N} \to A]$.

The answer is to work with *second-order* stream structures. It will be shown how to obtain functionals for (e.g.) recursion schemes for streams as ACPs on *second order* stream structures $\mathcal{S}^{(2)}$. It turns out that the simplest effective way to construct such a structure is to adjoin to the stream signature a level 2 functional $\boldsymbol{Sim}\colon (\mathbb{N} \to A) \to S$ which *simulates* every function $\varphi\colon \mathbb{N} \to A$ as a level 0 object $\boldsymbol{Sim}(\varphi) \in S$.

The next step is to extend the domain of $\boldsymbol{Sim}$ to include *potentially infinite* streams. As SF says: "These arise naturally both from mathematical computations and physical phenomena." An example of the first kind is obtained by *filtering* an infinite stream according a suitable condition on the items, where we may not know in advance whether the condition applies to finitely or infinitely many items in the stream. An example of the second kind is provided by irregularly received signals from an extraterrestial source, where we do not know at any point whether there will be any further signals.

Actually, the simplest and most elegant theory is obtained by allowing the domain of $\boldsymbol{Sim}$ to include all *partial functions* on $\mathbb{N}$, giving rise to "gappy" streams, from which the ACPs for potentially infinite streams can be obtained as a special case. This leads to second-order structures of the form

$$\mathcal{S} \;=\; (A,\, S,\; \boldsymbol{Cons},\, \boldsymbol{Hd},\; \boldsymbol{T\ell},\, \boldsymbol{Sim},\, \mathcal{N}) \tag{3.4}$$

where $\mathcal{N}$ is as in (3.2) and

(i) $A \neq \emptyset$,

[24] Note that this is the stream structure (2.5) without the equalities.

(ii) $\boldsymbol{Cons}\colon A \times S \to S, \quad \boldsymbol{Hd}\colon S \overset{\sim}{\to} A, \quad \boldsymbol{Tl}\colon S \to S, \quad \boldsymbol{Sim}\colon (\mathbb{N} \overset{\sim}{\to} A) \to S,$
(iii) $\forall a \in A \forall s \in S\big[\boldsymbol{Hd}(\boldsymbol{Cons}(a,s)) = a \ \wedge \ \boldsymbol{Tl}(\boldsymbol{Cons}(a.s)) = s\big],$
(iv) $\forall \varphi \in (\mathbb{N} \overset{\sim}{\to} A)\, \forall n \in \mathbb{N}\big[\boldsymbol{Hd}(\boldsymbol{Tl}^n(\boldsymbol{Sim}(\varphi))) \simeq \varphi(n)\big],$ and
(v) $s, s' \in S \ \wedge \ \forall n\big[\boldsymbol{Hd}(\boldsymbol{Tl}^n(s)) \simeq \boldsymbol{Hd}(\boldsymbol{Tl}^n(s'))\big] \implies s = s'.$

Let ***P-STREAM*** ("P" for "partial") be the ADT of all such structures.

This is the starting point for the analysis of computation on streams in Section 10. Since we are working with partial streams, a more refined concept of monotonicity is required than that given earlier in this paper (Section 2), namely *hereditary monotonicity*[25], requiring *chain-completeness*[26] of the partial orderings $\subseteq_i$ on all basic domains A_i.

Notation. For $\mathcal{S}$ as in (3.4), and $s \in S,\ n \in \mathbb{N},\ a \in A$, we write

- $(s)_n$ for $\boldsymbol{Hd}(\boldsymbol{Tl}^n(s))$,
- $\langle a; s\rangle$ for $\boldsymbol{Cons}(a,s)$, and
- $s^{\rightarrow}$ for $\boldsymbol{Tl}(s)$.

Then the ***partial ordering*** on the basic domains of $\mathcal{S}$ is defined by:

(i) $s \subseteq_S s'$ for $\forall n[(s)_n \downarrow \implies (s')_n \downarrow = (s)_n]$.
(ii) $\subseteq_A$ and $\subseteq_{\mathbb{N}}$ are equality on A and $\mathbb{N}$.

This makes all three basic orderings in $\mathcal{S}$ chain-complete.

Now consider the general situation of a structure $\mathcal{A}$ as in (3.1) where all the basic domains A_i $(i = 1, \dots, n)$ have chain-complete partial orderings $\subseteq_i$. We define, for certain function types σ, the domains, orderings, and concepts of monotonicity for that type.

First, taking $\sigma = (\bar{\imath} \overset{\sim}{\to} j)$, A_σ is the set of all $\varphi\colon A_{\bar{\imath}} \overset{\sim}{\to} A_j$ which are *monotonic*, in the sense that

$$\forall x, y \in A_{\bar{\imath}}, \big[\varphi(x) \downarrow \ \wedge \ x \subseteq_{\bar{\imath}} y \implies \varphi(y) \downarrow \ \wedge \ \varphi(x) \subseteq_j \varphi(y)\big]$$

(where the orderings $\subseteq_{i_k}$ on A_{i_k} are extended termwise to orderings $\subseteq_{\bar{\imath}}$ on $A_{\bar{\imath}}$ in the obvious way). Then the ordering $\subseteq_\sigma$ on A_σ is defined by: for all $\varphi, \psi \in A_\sigma$:

$$\varphi \subseteq_\sigma \psi \iff \forall x \in A_{\bar{\imath}}\big[\varphi(x) \downarrow \implies \psi(x) \downarrow \ \wedge \ \varphi(x) \subseteq_j \psi(x)\big]. \tag{3.5}$$

Next, *monotonicity* of level 2 functionals is defined by: $F\colon A_{\bar{\sigma}} \times A_{\bar{\imath}} \overset{\sim}{\to} A_j$ is monotonic if

$$\forall \varphi, \psi \in A_{\bar{\sigma}}\, \forall x, y \in A_{\bar{\imath}}\big[\, F(\varphi, x) \downarrow \ \wedge \ \varphi \subseteq_{\bar{\sigma}} \psi \ \wedge \ x \subseteq_{\bar{\imath}} y \implies F(\psi, y) \downarrow \ \wedge \ F(\varphi, x) \subseteq_j F(\psi, y)\big]. \tag{3.6}$$

Note that the basic function(al)s of $\mathcal{S}$ are all monotonic.

Then for a level 2 type $\tau = \sigma \times \bar{\imath} \overset{\sim}{\to} j$, we can define the domain A_τ of monotonic (partial) functionals of type τ, with the ordering[27]

$$F_1 \subseteq_\tau F_2 \iff \forall \varphi \in A_{\bar{\sigma}}\, \forall x \in A_{\bar{\imath}}\big[\, F_1(\varphi, x) \downarrow \implies F_2(\varphi, x) \downarrow \ \wedge \ F_1(\varphi, x) \subseteq_\tau F_2(\varphi, x)\big]. \tag{3.7}$$

[25] As in Platek's finite type structures [Fef77b] (cf. §1 above and footnote 3).
[26] I.e., any linearly ordered subset of A_i has a l.u.b in A_i.
[27] Not given explicitly in [Fef92b].

Note the similarity – and difference! – between the definitions given here ((3.5), (3.6), (3.7)) and those in [Fef92b] (§2 above): ((2.2), (2.3), (2.4)), where (essentially) it was assumed that the partial order on each basic domain is the identity.

Note also that the orderings defined in this way on these higher level domains are chain-complete (assuming the basic orderings are). So the technique used in §2 to construct LFPs and (hence) ACPs can be adapted to structures of the form (3.1) with chain-complete basic orderings and monotonic basic functionals F_i. This forms the basis of a *least fixed point* semantics for partial stream structures, using a version of the LFP Monotonicity Lemma of Section 3.

In particular, this theory can be applied to ***P-STREAM***, providing a justification for the *recursive schemes* for defining ACPs on partial stream structures given in Section 10, to which we turn below.

Discussion: Why a partial structure on streams?
There are two points here.

(1) We note above that the theory of LFPs is simpler when each basic domain has the identity as (trivial) partial order, as was done in §2. But that would make the basic functional ***Sim*** not monotonic.
(2) We *could* recover monotonicity for ***Sim*** by having *total streams* only in the stream domain S. However, this would complicate the theory of recursion schemes on streams. As SF says: "[T]he recursion schemata for partial streams (such as needed for the *Filter* operation) come out much more simply than they do for total streams."[28]

First, we note that the '***Sim***' functional characterizes stream structures up to *isomorphism*:

Categoricity Theorem for *P-STREAM* (Theorem 8).
Suppose structures $\mathcal{S} = (A, \dots)$ and $\mathcal{S}' = (A', \dots)$ both satisfy conditions (i)–(v). Then an isomorphism $A \cong A'$ can be extended to an isomorphism $\mathcal{S} \cong \mathcal{S}'$.

Now SF turns to the problem of a general formulation of stream recursion. He begins by stating: "By stream recursion we mean any general computational scheme for producing streams as values."

He does not attempt a single "most general" form (assuming that is even possible), but presents a number of schemes which have good practical applications, of which I'll give one example.

Note first that *partialness* (or "gappiness") of streams is sometimes a looser condition than we want. A more useful concept may be *potential infiniteness*, where a stream $s \in S$ is said to be potentially infinite (or "non-gappy") if

$$\forall n, m\big[(s)_n \downarrow \wedge\ m < n \implies (s)_m \downarrow\big].$$

We denote by S_{potinf} the subset of S consisting of these.

Let $\mathcal{S}^+ = (\mathcal{S}, \dots)$ be an expanded structure with $\mathcal{S} \in$ ***P-STREAM***.

Recursion Scheme (Theorem 10). Let C be a subset of one of the basic domains in $\mathcal{S}^+$. Given ACPs $G\colon C \to A$, $H_0, H_1\colon C \to C$ and $D\colon C \to \mathbb{B}$ over $\mathcal{S}^+$, we can find an ACP F over $\mathcal{S}^+$ satisfying

(i) $F\colon C \to S_{\mathsf{potinf}}$,
(ii) $F(c) = \big[\text{if } D(c) = \mathtt{tt} \text{ then } \langle G(c); F(H_0 c)\rangle \text{ else } F(H_1 c)\big]$ for all $c \in C$, and

[28] As we will see with the recursion scheme shown below.

(iii) if $F' : C \to S_{\mathsf{potinf}}$ is any function satisfying (ii) then $F(c) \subseteq F'(c)$ for all $c \in C$.

As SF points out[29]: "While F solves a fixed-point equation (ii), it cannot be described as its LFP, since that is the completely undefined function. Here, in contrast, F is total and is characterized by (iii) among all total solutions of (ii) as the one which is least pointwise in C."

An interesting application of this scheme is the *filtering operation* with respect to a predicate $\varphi : A \to \mathbb{B}$, where

$$\boldsymbol{Filter} : (A \to \mathbb{B}) \times S_{\mathsf{inf}} \ \to \ S_{\mathsf{potinf}}$$

is defined by

$$\boldsymbol{Filter}(\varphi, s) \ = \ \begin{cases} \langle (s)_0; \boldsymbol{Filter}(\varphi, s^{\to}) \rangle & \text{if } \ \varphi((s)_0) = \mathtt{tt} \\ \boldsymbol{Filter}(\varphi, s^{\to}) & \text{otherwise.} \end{cases}$$

This produces, from an infinite stream, a potentially infinite stream, which may or may not actually be finite ("extensionally" speaking).

We turn to Section 11, dealing with the recursion-theoretic interpretation of computation on stream structures over $\mathbb{N}$ – more precisely, structures for A-streams, where $A \subseteq \mathbb{N}$.

The *standard realization* for these takes $\mathcal{S}(A) = (\mathbb{N} \stackrel{\sim}{\to} A)$. In particular, the standard realization for $A = \mathbb{N}$ is

$$\mathcal{S}(\mathbb{N}) \ = \ (\mathbb{N},\, S(\mathbb{N}),\, \dots\,)$$

as in (3.4) with A replaced by $\mathbb{N}$, and $\boldsymbol{Sim}$ being the identity on $(\mathbb{N} \stackrel{\sim}{\to} \mathbb{N})$.

The substructure of $\mathcal{S}(\mathbb{N})$ induced by $A \subseteq \mathbb{N}$ is then

$$\mathcal{S}(A) \ = \ (A,\, S(A),\, \dots\,)$$

with $\boldsymbol{Sim}$ the identity on $(\mathbb{N} \stackrel{\sim}{\to} A)$.

By the Categoricity Theorem for $\boldsymbol{P\text{-}STREAM}$, every member $\mathcal{S}$ of $\boldsymbol{P\text{-}STREAM}$ on A has $\mathcal{S} \cong \mathcal{S}(A)$. Further, applying the Substructure Theorem in Section 4 to $\mathcal{S}(A)$, we have:

Substructure Theorem for $\mathcal{S}(A)$.
For each ACP F in $\Sigma(\boldsymbol{P\text{-}STREAM})$, $\mathcal{S}(A)$ is closed under $\mathsf{F}^{\mathcal{S}(\mathbb{N})}$, and

$$\mathsf{F}^{\mathcal{S}(\mathbb{N})} \restriction \mathcal{S}(A) \ = \ \mathsf{F}^{\mathcal{S}(A)}.$$

Thus, for any $A \subseteq \mathbb{N}$, a recursion-theoretic description of any ACP over $\mathcal{S}(A)$ is obtainable simply as the *restriction* of that ACP over $\mathcal{S}(\mathbb{N})$. To clarify this: replace $\mathcal{S}(\mathbb{N})$ by the structure

$$\mathcal{E}(\mathbb{N}) \ = \ (\mathbb{N},\, S(\mathbb{N}),\, \boldsymbol{Eval},\, \boldsymbol{Sim},\, \boldsymbol{Sc},\, \boldsymbol{Pd},\, \mathbf{0},\, \boldsymbol{Eq_0}).$$

where $\boldsymbol{Eval} : S(\mathbb{N}) \times \mathbb{N} \ \stackrel{\sim}{\to} \ \mathbb{N}$ is given by $\boldsymbol{Eval}(s, n) \simeq s(n)$. This structure is easily seen to be equivalent to $\mathcal{S}(\mathbb{N})$, in the sense that every ACP over $\mathcal{S}(\mathbb{N})$ is obtainable as one over $\mathcal{E}(\mathbb{N})$, and conversely.

Note that $\mathcal{E}(\mathbb{N})$ has two basic domains: $A_0 = \mathbb{N}$ and $A_1 = S(\mathbb{N})$, of type levels 0 and 1 respectively. This suggests a straightforward interpretation of $\mathcal{E}(\mathbb{N})$ into the finite type structure over $\mathcal{N}$ (cf. (3.2)), which contains only one basic domain, $A_0 = \mathbb{N}$ of level 0. By this interpretation, monotonic partial functionals over $\mathcal{S}(\mathbb{N})$

[29] In connection with another recursion scheme, but it is still appropriate here.

(or $\mathcal{E}(\mathbb{N})$) of type level 2 (or less) are identified with monotonic partial functionals of type level 3 (or less) over $\mathcal{N}$.

Hence, for a recursion-theoretic interpretation of ACPs over $\mathcal{S}(\mathbb{N})$, we need an extension of the notion of *partial recursiveness* to functionals of type level 3 over $\mathcal{N}$.

This was provided by Ershov [Ers72] (as outlined in §1 above) in a structure $C^\smile = (C_n^\smile)_n$ with a notion of *partial recursiveness* for functionals of arbitrary finite type, with hereditarily partial continuous arguments over $C_0^\smile = \mathbb{N}$, based on an abstract theory of f-spaces (a special kind of topological space).

This theory was then simplified by SF [Fef77b, Fef77a] to a "concrete" theory $\boldsymbol{PR}/C^\smile$ of *partial recursive continuous functionals* of finite type, analogous to functionals of hereditarily *total* continuous arguments [Kle59a, Kre59], using (again) a system of *formal neighbourhoods*. The precise definitions are given in the paper. The following result can then be obtained via Ershov's theory of f-spaces [Ers72]. A simpler, direct proof[30] is possible via SF's version of the theory indicated above.

Theorem (Closure of $\boldsymbol{PR}/C^\smile$ under ACP).
$\boldsymbol{PR}/C^\smile$ is closed under the extension of ACP schemata to arbitrary finite types.

It follows that ACPs over $\mathcal{S}(\mathbb{N})$ of level ≤ 2 can be re-interpreted as *partial recursive continuous functionals* over $\mathcal{N}$ of level ≤ 3.

The paper has three appendices. I will only remark on *Appendix B: Comparison with the work of Tucker and Zucker*. This concerns research that John Tucker and I have done in a series of publications[31] on many-sorted models of computation $\mathcal{A}$. One model over $\mathcal{A}$ that we have investigated is $\boldsymbol{\mu PR}^*(\mathcal{A})$, consisting of schemes for primitive recursion over $\mathcal{A}^*$ plus μ (the "constructive least number operator"), where $\mathcal{A}^*$ is formed by adding, to each carrier set A of $\mathcal{A}$, a set A^* of all finite sequences from A (with associated basic operations). Note that to formulate this model, we must assume that $\mathcal{A}$ includes, as a subalgebra, the algebra $\mathcal{N}$ of naturals — or add it on. This model also forms a basis for a *Generalized Church-Turing Thesis*[32].

SF noted that for first-order functions on $\mathcal{A}$, $\boldsymbol{\mu PR}^*(\mathcal{A}) \subseteq \mathbf{ACP}(\mathcal{A}^*)$. He conjectured the reverse inclusion, leading to the equality

$$\boldsymbol{\mu PR}^*(\mathcal{A}) \;=\; \mathbf{ACP}(\mathcal{A}^*).$$

This was proved in [XZ04], assuming a modification of SF's LFP schemes (cf. §2 above) formed by replacing the simple LFP scheme (VIII) by a *simultaneous* LFP. This is discussed further in the last of our four papers by SF (§4 below), to which we now turn.

4. About and around computing over the reals

An overview of this paper is provided by the first section, in which SF sets the stage by referring to a "very interesting and readable" article by Lenore Blum [Blu04], which explains the so-called BSS model of computation over the reals due to Blum, Shub and Smale [BSS89], expounded also in the well-known book by Blum, Cucker, Shub and Smale [BCSS98].

---

[30] Unfortunately never published (personal communication by SF).
[31] See e.g. [TZ15] and the references therein.
[32] Discussed further in §4 below.

Blum claimed that the BSS model of computation on reals is the appropriate foundation for *scientific computing*.

Braverman and Cooke [BC06] argued rather for a *bit computation* model, *prima facie* incompatible with the BSS model. This goes back to ideas of Banach and Mazur in the 1930s, improved by Grzegorczyk and Lacombe (independently, in the 1950s). SF proposes rather to name these "*effective approximation*" models of computation. Later in this paper he discusses such models further (see below).

We should note that there are functions computable in each of these two models which are not computable in the other.

We note also that the bit-computable model only computes *continuous* functions. I consider this a positive rather than a negative property of the model, in keeping with the *continuity principle*:

$$\textit{computability} \implies \textit{continuity}. \tag{4.1}$$

This is related to Hadamard's principle [Had52] which, as (re-)formulated by Courant and Hilbert ([CH53, pp. 227ff.],[Had64]) states that for a scientific problem to be well posed, the solution must (apart from existing and being unique) depend continuously on the data.[33]

On the topic of comparing models, I now quote SF extensively. He asks:

> Despite their incompatibility, is there any way that these can both be considered to be reasonable candidates for computation on the real numbers?

— and gives an "obvious answer":

> [T]he BSS model may be considered to be given in terms of computation over the reals as an *algebraic structure*, while ... the effective approximation model can be given ... as a *topological structure* of a particular kind, or alternatively as a *second order structure over* the rationals. But all such explanations presume a general theory of *computation over an arbitrary structure*.

He continues:

> After explaining the BSS and effective [approximation] models respectively in sections 2 and 3 below, my main purpose here is to describe three theories of computation over (more or less) arbitrary strutures in sections 4 and 5, the first due to Harvey Friedman, the second due to John Tucker and Jeffery Zucker, and the third due to the author, adapting to earlier work of Richard Platek and Yannis Moschovakis. Finally, and in part as an aside, I shall relate the effective approximation approach to the foundations of constructive analysis in its groundbreaking form due to Errett Bishop.

SF concludes the Introduction by touching on the relevance of these structures to *scientific computation* (or "*numerical analysis*"):

> The justification for particular techniques varies with the areas of application but there are common themes that have to do with identifying the source and control of errors and with efficiency of computation. However, there is no concern in the literature on scientific computation with the underlying nature of computing with the reals as exact objects. For, in practice, those computations are made in "floating point arithmetic"

[33] These issues are discussed in [TZ15, §7],[TZ11, §4.2.14].

> using finite decimals with relatiely few significant digits, for which computation *per se* reduces to computation with rational numbers.

He also notes:

> Besides offering a theory of computation on the real numbers, the main emphasis in the articles [Blu04, BC06] and the book [BCSS98] is on the relevance to the subject of scientific computation in terms of *measures of complexity* ... While complexity issues must certainly be taken into account in choosing between the various theories of computation over the reals on offer as a foundation for scientific computation, I take no position as to which of these is most appropriate for that purpose. Rather, my main aim here is to compare them on *purely conceptual grounds*.[34]

Section 2 provides a quick survey of ***BSS-type models***. SF refers to [Blu04], "a brief but informative description", and to the more detailed description in [BCSS98].

The BSS definition makes sense for any ring or field, including $\mathbb{R}$, $\mathbb{C}$, $\mathbb{R}^n$, etc., making it an "*algebraic* conception of computability":

> This is reflected in the fact that inputs to a machine for computing a given algorithm are *unanalyzed entities* in the algebra A, and that a basic admitted step in a computation procedure is to test *whether two machine contents x and y are equal or not* ... [and] in case A is ordered, ... *whether $x < y$ or not*.[35]

There are finite and infinite dimensional versions. An example of an algorithm in this formalism in the finite-dimenional case is the Newton algorithmn for $\mathbb{R}$ or $\mathbb{C}$. An example for the infinite-dimensional case is testing whether a finite set of polynomials over $\mathbb{C}$ have a common zero; this is related to the Hilbert Nullstellensatz.

It is pointed out in [BCSS98] that in the finite-dimensional case, a BSS algorithm can be implemented as a form of register machine, and in the infinite-dimensional case, as a form of Turing machine with 2-way infinite tapes. In the case of rings and fields, only piecewise polynomial and rational functions (respectively) are computable.

> In the opposite direction, one may ask what BSS algorithms can actually be carried out on a computer. Here Tarski's decision procedure for the algebra of real numbers is relevant, as it reduces a question of the Hilbert Nullstellensatz type, concerning common roots of a set of polynomials, to a quantifier-free condition on the their coefficients. On the face of it, Tarski's procedure runs in time complexity as a tower of exponentials.

This can be improved to doubly exponential upper bounds by the method of cylindrical algebra decomposition [CJ98].

In ***Section 3***, SF turns to ***effective approximation*** (***EA***) ***models***. We consider functions $f : \mathbb{I} \to \mathbb{R}$, where $\mathbb{I}$ is a (finite or infinite) real interval.

There are two main approaches here:

(1) S-effective approximation: working with sequencess of approximating arguments and values;

---

[34] Emphasis added. Following SF, I shall focus on the *real-computability* aspect of the various models, rather than their complexity-theoretical or scientific-computational aspects.

[35] Emphasis added.

(2) P-effective approximation: approximating functions by polynomials.

To illustrate (1): Suppose $f(x) = y$. We work with a sequential representation of x, i.e., a Cauchy sequence of rationals $\langle q_n \rangle$ with limit x, to effectively determine a Cauchy sequence $\langle r_n \rangle$ of rationals with limit y. We may also assume the Cauchy sequences are "fast", i.e., $|q_n - x| \leq 2^{-n}$. With the sequences $\langle q_n \rangle$ and $\langle r_n \rangle$ coded as functions $\varphi, \psi \colon \mathbb{N} \to \mathbb{N}$ in a standard way, the S-effective approximation computability of f reduces to finding an *effectively computable* functional $F \colon \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ of type level 2 over $\mathbb{N}$.

Writing $\mathcal{P}$ and $\mathcal{T}$ for the classes of partial and total functions (respectively) from $\mathbb{N}$ to $\mathbb{N}$: for "*effective computability*" of functionals $F \colon \mathcal{P} \to \mathcal{P}$ we can use Kleene's notion of *partial recursiveness* of functionals – as has already been done above in all three previous papers.[36] This definition ensures that F is monotonic and continuous on $\mathcal{P}$, and hence, as SF writes, "computable functionals $F \colon \mathcal{T} \to \mathcal{T}$ may be defined as those partial recursive functionals whose value for each total function φ is a total function $F(\varphi)$."

SF continues: "[T]here are several other ways of defining which are the computable functionals $F \colon \mathcal{T} \to \mathcal{T}$ without appealing to the notion of partial recursive functionals." For example, Grzegorczyk [Grz55] used a generalization of Kleene's schemata [Kle52] for general recursive functions with both primitive recursion and the least number operator.

In fact this notion of computable real function was shown by Grzegorczyk [Grz57] to be equivalent to one formulated by him and Lacombe [Lac55] independently. In the simple version stated in [PER89], this says (roughly) that a function from the reals to the reals is computable if (i) it maps computable sequences of points to computable sequences of points; and (ii) it satisfies an effective locally uniform continuity condition. We call this *Grzegorczyk/Lacombe* (*GL*) computability. This turns out to be equivalent to Weihrauch's Type-2 Theory of Effectivity (TTE) [Wei00]. To quote SF:

> In my view, Kleene's notion of partial recursive functional is the fundamental one, in that it specializes to Grzegorczyk's (or Weihrauch's) in the following sense: if F is a partial recursive functional $F \colon \mathcal{P} \to \mathcal{P}$, and $F|\mathcal{T}$, the restriction of F to $\mathcal{T}$, maps $\mathcal{T}$ to $\mathcal{T}$, then $F|\mathcal{T}$ is definable by the Grzegorczyk schemata, as may easily be shown.

SF continues:

> It is a consequence of the continuity of partial recursive functionals that if F effectively represents a real-valued function f on its domain $\mathbb{I}$, then f is continuous at each point of $\mathbb{I}$. ... Thus, unlike the BSS model, the order relation on $\mathbb{R}$ is not computable.

To formulate essentially the same phenomenon differently: In all versions of the S-effective and P-effective computation theories on the reals considered here (GL, Weihrauch, etc., and see the next section), and in contrast to the BSS model, order and equality on the reals is *not* computable – specifically, equality on $\mathbb{R}$ is co-semicomputable.

In ***Section 4***: ***The view from generalized recursion theory*** (***g.r.t.***) SF considers two generalizations of recursion theory to arbitrary structures.

[36] See footnote 4.

First, there is Harvey Friedman's adaptation of the register machine approach [Fri71]. He dealt with structures of the form

$$\mathcal{A} \;=\; (A, c_1, \ldots, c_j, f_1, \ldots, f_k, R_1, \ldots, R_m)$$

Comparing this to the stucture in (3.1) above, we note that (unlike the latter) there is here only one domain A (to be changed later); and further (as in (3.1)) equality is not necessarily assumed as a basic operation on A.

A *finite algorithmic procedure* (*fap*) π on $\mathcal{A}$ is given by a finite list of instructions $I_1, \ldots, I_t$. There are also register names $r_0, r_1, r_2, \ldots$ (functioning as variables), with r_0 reserved for output. The instructions include assignments to the r_i and branching on a conditional given by one of the R_i. The class of fap computable functions is denoted by $\mathbf{FAP}(\mathcal{A})$.

For the structure $\mathcal{N}$ of naturals (as in (3.2)) $\mathbf{FAP}(\mathcal{N})$ is equal to the partial recursive functions.

This notion can be generalized to many-sorted structures $\mathcal{A}$ as in (3.1).

Friedman also introduced the class $\mathbf{FAPC}(\mathcal{A})$ (*faps with counting over* $\mathcal{A}$), corresponding to $\mathbf{FAP}(\mathcal{A}, \mathcal{N})$, where $(\mathcal{A}, \mathcal{N})$ denotes $\mathcal{A}$ augmented by $\mathcal{N}$.

A further extension was made by Moldestad, Stoltenberg-Hansen and Tucker [MSHT80b, MSHT80a] to incorporate *stack registers* in the model, producing the structure $\mathbf{FAPS}(\mathcal{A})$ of *faps with stacks over* $\mathcal{A}$, and $\mathbf{FAPCS}(\mathcal{A})$ of *faps with counting and stacks over* $\mathcal{A}$.

To bring this into the realm of the main concern of this article, consider the structure of the reals

$$\mathcal{R} \;=\; (\,\mathbb{R},\, 0,\, 1,\, +,\, -,\, \times,\, ^{-1},\, =,\, <\,)$$

(with decidable equality and order). Friedman and Mansfield [FM92] showed the equivalence of $\mathbf{FAPS}(\mathcal{R})$ to the BSS model.

SF also discusses my collaborative work with John Tucker (see, e.g., the lengthy survey paper [TZ00] or the more recent [TZ15]) dealing with a high level 'while' programming language over abstract many-sorted algebras [37]. It is assumed that such an algebra $\mathcal{A}$, of sort Σ, is *standard* in the sense of containing the sort of booleans, with the standard boolean operations.

We also consider expansions of $\mathcal{A}$: $\mathcal{A}^N$, which includes the algebra $\mathcal{N}$ of naturals, and $\mathcal{A}^*$, which includes (further) for each basic domain A_i of $\mathcal{A}$, also a domain A_i^* of finite sequences of elements of A, with associated basic operations, having signatures Σ^N and Σ^* respectively. For a signature Σ of such an algebra, we consider the class of $\boldsymbol{While}(\Sigma)$ program statements generated by:

$$S ::= \mathsf{skip} \mid \mathsf{x} := t \mid S_1 \,;S_2 \mid \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi} \mid \mathsf{while}\ b\ \mathsf{do}\ S_0\ \mathsf{od}$$

where the variable x and term t have the same Σ-sort. Then $\boldsymbol{While}(\mathcal{A})$, $\boldsymbol{While}^N(\mathcal{A})$ and $\boldsymbol{While}^*(\mathcal{A})$ are the classes of (partial) functions on $\mathcal{A}$ definable (respectively) by $\boldsymbol{While}(\Sigma)$, $\boldsymbol{While}(\Sigma^N)$ and $\boldsymbol{While}(\Sigma^*)$ procedures. It follows from

[37] We use "algebra" rather than "structure" to indicate that their signatures contain function (and constant) but not relation symbols.

[MSHT80a, TZ00] that for any standard algebra $\mathcal{A}$,

$$\begin{aligned} \textbf{\textit{While}}(\mathcal{A}) &= \mathbf{FAP}(\mathcal{A}) \\ \textbf{\textit{While}}^N(\mathcal{A}) &= \mathbf{FAPC}(\mathcal{A}) \\ \textbf{\textit{While}}^*(\mathcal{A}) &= \mathbf{FAPCS}(\mathcal{A}). \end{aligned}$$

On the basis of this and other results, John Tucker and I have presented a *generalized Church-Turing thesis for algebraic computability* on standard many-sorted algebras $\mathcal{A}$ involving $\textbf{\textit{While}}^*$ computability on $\mathcal{A}$ [TZ15].

Remark (Compatibility between algebraic and EA models).
Recall the two approaches to computability on the reals signalled by SF at the beginning of this article: *algebraic* (as exemplified by the BSS model) and *effective approximability* (EA) (e.g., Grzegorczyk-Lacomb or Weierstrass). Our $\textbf{\textit{While}}^*$ model would fall into the "algebraic" class. Let us call this approach to computability on the reals "abstract", and the EA approach "concrete". There would seem to be an *incompatibility* between the abstract and concrete approaches, since in e.g. the BSS model, but *not* in the topological models, equality and order on the reals are total and (hence) not continuous, but nonetheless computable. In the EA (but not the BSS) models, equality and order are given as *partial* (boolean-valued) operations, which are continuous, and also computable. This is in accordance with the continuity principle discussed above (cf. (4.1)).

The point here is that by considering *topological partial algebras* on the reals, in which the basic operations may be partial and are all continuous, we can recover *compatibility between abstract and concrete models*.

In fact, in [TZ04] we proved the equivalence of *abstract* (*computable*) *approximability* by $\textbf{\textit{While}}^*$ programs augmented by a "countable choice" operator, and *concrete computability*, on *metric partial algebras*, under restrictions of effective locally uniform continuity. This result applies, for example, to algebras such as a partial version $\mathcal{R}_\mathsf{p}$ of $\mathcal{R}$.

In ***Section 5: The higher type approach***, SF applies his theory $\mathbf{ACP}(\mathcal{A})$ developed in [Fef92b, Fef96] (§§2,3 above) of *abstract computation procedures* over many-sorted higher order algebras $\mathcal{A}$ to the special case $\mathcal{A} = \mathcal{N}$. Recall that these ACPs are monotonic, continuous, partial functionals generated by schemata, including (notably) a schema for the least fixed point functional. Further, the sets $\mathbf{ACP}^1(\mathcal{N})$ and $\mathbf{ACP}^2(\mathcal{N})$ of ACPs of levels 1 and 2 over $\mathcal{N}$ correspond to Kleene's partial recursive functions and functionals at these levels.[38]

The interesting point here is that the reals are taken, not as elements of one of the basic domains A_i, but as functions of type level 1, $f\colon \mathbb{N} \to \mathbb{N}$, representing effective Cauchy sequences of rationals (under suitable coding).

Let us write ***Rep*** for the class of such functions $f, g, \ldots$. Under this representation, the computable functions over the reals can be identified with those functionals in $\mathbf{ACP}^2(\mathcal{N})$ which map ***Rep*** to ***Rep***, preserving the '$\equiv$' relation on ***Rep***, where $f \equiv g$ means that their corresponding Cauchy sequences have the same limit. To quote SF again:

> So now the S-approximation theory of effective computability of functions of real numbers is explained essentially as in sec. 3 above in terms

[38] See footnote 4.

> of total recursive functionals in $\mathbf{ACP}^2(\mathcal{N})$. ... Thus abstract computation procedures provide another way of subsuming the two approaches to computation over the real numbers at a basic conceptual level. Of course, this in no way adjudicates the dispute over the proper way to found scientific computation on the real numbers or to deal with the relevent questions of complexity.

SF makes one more important point in this section: how the above illustrates the difference between *extensional* and *intensional* aspects of computation.

> On the face of it, the BSS approach is extensional, while that of S-effective approximation theory is intensional in its essential use of ***Rep*** and $\equiv$ on ***Rep***. But there is an even more basic difference ... Namely, functions $f, g, h, \ldots$ there are tacitly understood in the usual set-theoretic sense for which the extensionality principle ... holds, i.e., if $f(n) = g(n)$ for all n in $\mathbb{N}$, then $f = g$. By the *intensional recursion-theoretic interpretation* of $\mathbf{ACP}(\mathcal{N})$ I mean what one gets by taking the function variables $f, g, h, \ldots$ to range instead over *indices* of partial recursive functions .... Now one proves inductively for this interpretation that each F in $\mathbf{ACP}^2(\mathcal{N})$ preserves extensional equality and hence is an effective operator in the sense of Myhill and Shepherdson (1955), i.e., if $f \equiv g$ then $F(f) \equiv F(g)$. .... In the end, when speaking about actual computation, we have intensionality throughout, since computers only work with finite symbolic representations of the objects being manipulated.

Finally, ***Section 6: the Bishop approach to constructive analysis*** is interesting, in that it is the only approach to computing on the reals discussed in this paper which is based on an *informal* notion of computation. This comes from an investigation into the concept of *constructive analysis* carried out by Errett Bishop in his book [Bis67] and his book with Douglas Bridges [BB85]. SF gives a good summary of the philosophical and technical aspects of this program, the details of which I omit here.

Briefly, the point of Bishop's constructive mathematics (hereinafter BCM) is that existential assertions must produce witnesses of an existential claim. These witnesses then provide a constructive function of the other parameters of the problem. The problem, however, is (to quote SF again *in extenso*):

> What is not clear from Bishop's [Bis67] or that of Bishop and Bridges [BB85] is how the computational content of the results obtained is to be accounted for in recursion-theoretic terms, in the sense of ordinary or generalized recursion theory as discussed in secs. 3–5 above. From the logical point of view, this may be accomplished by formalizing the work of [BB85] (and BCM more generally) in a formal system T that has recursive interpretations. A variety of such systems were proposed in the 1970s, first by Bishop himself and then by Nicholas Goodman, Per Martin-Löf, John Myhill, Harvey Friedman and me, and surveyed in [Fef79] ... (cf. also [Bee85]).

About these formal systems, SF continues:

> Roughly speaking, those account for the computational content of [Bis67] in two different ways: the first treats witnessing information implicitly and depends for its extraction on the fact that the systems are formalized in intuitionistic logic, while the second kind treats witnessing information explicitly as part of the package explaining each notion and does not require the logic to be intuitionistc. For the first kind of system,

> the method of extraction is by ... the method of recursive realizability introduced by Kleene or by the use of (recursive) functional interpretations originated by Gödel. Only the system T_0 of Explicit Mathematics introduced in [Fef75], and applied to BCM in [Fef79] is of the second kind ...

I omit a description of T_0 except to say that it has variables of two kinds, for individuals and classes, and the basic relation between individuals, besides identity, is the 3-place relation $\boldsymbol{App}(x, y, z)$, with the meaning $\{x\}y \simeq z$ in ordinary recursion theory.

Then case studies of typical arguments in BCM show that it can be formalized in a subsystem of T_0 of the same strength as Peano Arithmetic [Fef79]; in fact, work of Feng Ye [Ye00] suggests that this can already be done in a subsystem of the strength of Primitive Recursive Arithmetic.

Turning finally to the issue of *feasibility*, let SF have the last word on BCM:

> [T]he practice of Bishop style constructive analysis needs to be examined directly for turning its results that predict computability in principle to ones that demonstrate computability in practice. Presumably all of the specific methods of scientific computation are subsumed under Bishop style constructive mathematics. Assuming that is the case, here is where a genuine connection might be made between constructive mathematics, the theory of computation, and scientific computation, which puts questions of complexity up front.

Conclusion

This brings to a close my view of SF's groundbreaking work in generalized computability theory, by means of a close look at four of his papers through the years.

Lack of time and space have prevented discussion of more papers. Let me at least recommend one of his last papers [Fef15], in which he considered various proposals for generalizing the Church-Turing Thesis to concrete and abstract structures.

References

[BB85] E. Bishop and D. Bridges. *Constructive Analysis*. Springer-Verlag, 1985.

[BC06] M. Braverman and S. Cook. Computing over the reals: foundations for scientific computation. *Notices of the American Mathematical Society*, 51:318–329, 2006.

[BC13] P. Bürgisser and F. Cucker. *Condition: The Geometry of Numerical Algorithms*. Springer, 2013.

[BCSS98] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer, 1998.

[Bee85] M. Beeson. *Foundations of Constructive Mathematics*. Springer-Verlag, 1985.

[Bis67] E. Bishop. *Foundations of Constructive Analysis*. McGraw-Hill, 1967.

[Blu04] L. Blum. Computability over the reals: Where Turing meets Newton. *Notices of the American Mathematical Society*, 51:1024–1034, 2004.

[BSS89] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21:1–46, 1989.

[BT83] J.A. Bergstra and J.V. Tucker. Initial and final algebra semantics for data type specifications: two characterization theorems. *SIAM Journal of Computing*, 12:366–387, 1983.

[CH53] R. Courant and D. Hilbert. *Methods of Mathematical Physics, Vol. II*. Interscience, 1953. Translated and revised from the German edition [1937].

[CJ98] B.F. Caviness and J.R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer, 1998.

[Ers72] Y.L. Ershov. Computable functions of finite types. *Algebra and Logic*, 11:203–242, 1972. Translated from *Algebra i Logika* 11:367–437, 1972.

[Ers74] Y.L. Ershov. Maximal and everywhere-defined funtionals. *Algebra and Logic*, 13:210–225, 1974. Translated from *Algebra i Logika* 13:374–397, 1974.

[Fef75] S. Feferman. A language and axioms for explicit mathematics. In J.N. Crossley, editor, *Algebra and Logic: Papers from the 1974 Summer Research Institute of the Australasian Mathematical Society*, volume 450 of *Lecture Notes in Mathematics*, pages 87–139. Springer, 1975.

[Fef77a] S. Feferman. Generating schemes for partial recursively continuous functionals (Summary). In *Colloque International de Logique, Clermont-Ferrand, 1975*, volume 249 of *Colloq. Int. du Centre National de la Recherche Scientifique, Paris*, pages 191–198, 1977.

[Fef77b] S. Feferman. Inductive schemata and recursively continuous functionals. In R.O. Gandy and J.M.E. Hyland, editors, *Logic Colloquium 76: Proceedings of a Conference held in Oxford in July 1976*, pages 191–198. North Holland, 1977.

[Fef79] S. Feferman. Constructive theories of functions and classes. In M. Boffa, D. van Dalen, and K. McAloon, editors, *Logic Colloquium '78: Proceedings of a Colloquium held in Mons, August 1978*, pages 159–224. North Holland, 1979.

[Fef92a] S. Feferman. A new approach to abstract data types, I: Informal development. *Mathematical Structures in Computer Science*, 2:193–229, 1992.

[Fef92b] S. Feferman. A new approach to abstract data types, II: Computability on adts as ordinary computation. In E. Börger *et al.*, editor, *Computer Science Logic: Proc. 5th Workshop, Berne, Switzerland, Oct. 1991*, volume 626 of *Lecture Notes in Computer Science*, pages 79–95. Springer-Verlag, 1992.

[Fef96] S. Feferman. Computation on abstract data types: The extensional approach, with an application to streams. *Annals of Pure & Applied Logic*, 81:75–113, 1996.

[Fef13] S. Feferman. About and around computing over the reals. In B.J. Copeland, C.J. Posy, and O. Shagrir, editors, *Computability: Turing, Gödel, Church, and Beyond*, pages 55–76. MIT Press, 2013.

[Fef15] S. Feferman. Theses for computation and recursion on abstract structure. In G. Sommaruga and T. Strahm, editors, *Turing's Revolution: The Impact of his Ideas about Computability*. Birkhäuser/Springer Basel, 2015.

[Fen74] J.E. Fenstad. On axiomatizing recursion theory. In J.E. Fenstad and P.G. Hinman, editors, *Generalized Recursion Theory: Proc. 1972 Oslo Symp.*, pages 385–404. North Holland, 1974.

[FM92] H. Friedman and R. Mansfield. Algebraic procedures. *Transactions of the American Mathematical Society*, 332:297–312, 1992.

[Fri71] H. Friedman. Algebraic procedures, generalized Turing algorithms, and elementary recursion theory. In R.O. Gandy and C.M.E. Yates, editors, *Logic Colloquium '69*, pages 361–389. North Holland, 1971.

[Grz55] A. Grzegorczyk. Computable functions. *Fundamenta Mathematicae*, 42:168–202, 1955.

[Grz57] A. Grzegorczyk. On the defintions of computable real continuous functions. *Fundamenta Mathematicae*, 44:61–71, 1957.

[GTWW77] J.A. Goguen, J.W. Thatcher, E.G. Wagner, and J.B. Wright. Initial algebra semantics and continuous algebras. *Journal of the Association for Computing Machinery*, 24:68–95, 1977.

[Had52] Jacques Hadamard. *Lectures on Cauchy's Problem in Linear Partial Differential Equations*. Dover, 1952. Translated from the French edition [1922].

[Had64] J. Hadamard. *La Théorie des Équations aux Dérivées Partielles*. Éditions Scientifiques, 1964.

[Hyl75] J.M.E. Hyland. *Recursion Theory on the Conntable Functionals*. D.Phil. thesis, Oxford University, 1975.

[Kle52] S.C. Kleene. *Introduction to Metamathematics*. North Holland, 1952.

[Kle59a] S.C. Kleene. Countable functionals. In A. Heyting, editor, *Constructivity in Mathematics: Proceedings of the Colloquium held at Amsterdam, 1957*, pages 81–100. North Holland, 1959.

[Kle59b] S.C. Kleene. Recursive functionals and quantifiers of finite types i. *Transactions of the American Mathematical Society*, 91:1–52, 1959.

[KM77] A.S. Kechris and Y.N. Moschovakis. Recursion in higher types. In J. Barwise, editor, *Handbook of mathematical logic*, pages 681–737. North Holland, 1977.

[Kre59] G. Kreisel. Interpretation of analysis by means of constructive functionals of finite type. In A. Heyting, editor, *Constructivity in Mathematics: Proceedings of the Colloquium held at Amsterdam, 1957*, pages 101–128. North Holland, 1959.

[Lac55] D. Lacombe. *Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles*, I, II, III. *C.R. Acad. Sci. Paris*, 1955. 240:2470–2480, 241:13–14,151–153.

[Mos69] Y.N. Moschovakis. Abstract first order computability I. *Transactions of the American Mathematical Society*, 138:427–464, 1969.

[Mos71] Y.N. Moschovakis. Axioms for computation theories — first draft. In R.O. Gandy and C.E.M. Yates, editors, *Logic Colloquium '69: Proc. Summer School & Colloq. in Math. Logic, Manchester, Aug. 1969*, pages 199–255. North Holland, 1971.

[Mos74] Y.N. Moschovakis. *Elementary Induction on Abstract Structures*. North Holland, 1974.

[Mos76] Y.N. Moschovakis. On the basic notions in the theory of induction. In *Proc. 5th International Congress in Logic, Methodology and Philosophy of Science, London, Ontario, 1975*, pages 207–236, 1976.

[Mos84] Y.N. Moschovakis. Abstract recursion as a foundation for the theory of recursive algorithms. In M.M. Richter *et al.*, editor, *Computation and Proof Theory: Proc. Logic Colloquium Aachen 1983, Part II*, volume 1104 of *Lecture Notes in Mathematics*, pages 289–364. Springer-Verlag, 1984.

[Mos89] Y.N. Moschovakis. The formal language of recursion. *Journal of Symbolic Logic*, 54:1216–1252, 1989.

[MP85] J.C. Mitchell and G.D. Plotkin. Abstract types have extensional type. *ACM Trans. Progr. Languages & Systems*, 10:470–502, 1985.

[MS55] J. Myhill and J. Shepherdson. Effective operations on partial recursive functions. *Zeitschr. Msth. Logik u. Grundlagen. Math.*, 1:310–317, 1955.

[MSHT80a] J. Moldestat, V. Stoltenberg-Hansen, and J.V. Tucker. Finite algorithmic procedures and computation theories. *Mathematica Scandinavica*, 46:77–94, 1980.

[MSHT80b] J. Moldestat, V. Stoltenberg-Hansen, and J.V. Tucker. Finite algorithmic procedures and inductive definability. *Mathematica Scandinavica*, 46:62–76, 1980.

[PER89] M.B. Pour-El and J.I. Richards. *Computability in Analysis and Physics*. Springer-Verlag, 1989.

[Pla66] R.A. Platek. *Foundations of Recursion Theory*. Ph.D. thesis, Department of Mathematics, Stanford University, 1966.

[Saz76] V.Y. Sazonov. Degrees of parallelism in computation. In A. Mazurkiewicz, editor, *Proc. 5th Symp. Mathematical Foundations of Computer Science (MFCS'76), Gdansk, Sept. 1976*, volume 45 of *Lecture Notes in Computer Science*, pages 517–523. Springer-Verlag, 1976.

[Smu61] R.M. Smullyan. *Theory of Formal Systems*, volume 47 of *Annals of Mathematical Studies*. Princeton University Press, 1961.

[Str71] H.R. Strong, Jr. Translating recursion equations into flowcharts. *Journal of Computer & Systems Science*, 5:254–285, 1971.

[TZ88] J.V. Tucker and J.I. Zucker. *Program Correctness over Abstract Data Types, with Error-State Semantics*, volume 6 of *CWI Monographs*. North Holland, 1988.

[TZ00] J.V. Tucker and J.I. Zucker. Computable functions and semicomputable sets on many-sorted algebras. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 5, pages 317–523. Oxford University Press, 2000.

[TZ04] J.V. Tucker and J.I. Zucker. Abstract versus concrete computation on metric partial algebras. *ACM Transactions on Computational Logic*, 5:611–668, 2004.

[TZ11] J.V. Tucker and J.I. Zucker. Continuity of operators on continuous and discrete time streams. *Theoretical Computer Science*, 412:3378–3403, 2011.

[TZ15] J.V. Tucker and J.I. Zucker. Generalizing computability to abstract algebras. In G. Sommaruga and T. Strahm, editors, *Turing's Revolution: The Impact of his Ideas about Computability*. Birkhauser/Springer Basel, 2015.

[Wei00] K. Weihrauch. *Computable Analysis: An Introduction*. Springer, 2000.

[XZ04] Jian Xu and Jeffery Zucker. First and second order recursion on abstract data types. *Fundamenta Informaticae*, 21:1–43, 2004.

[Ye00] Feng Ye. Toward a constructive theory of unbounded linear operators. *Journal of Symbolic Logic*, 65:357 – 370, 2000.