About and Around Computing Over the Reals

Solomon Feferman

1. One theory or many? In 2004 a very interesting and readable article by Lenore Blum, entitled "Computing over the reals: Where Turing meets Newton," appeared in the *Notices of the American Mathematical Society*. It explained a basic model of computation over the reals due to Blum, Michael Shub and Steve Smale (1989), subsequently exposited at length in their influential book, *Complexity and Real Computation* (1997), coauthored with Felipe Cucker. The 'Turing' in the title of Blum's article refers of course to Alan Turing, famous for his explication of the notion of mechanical computation on discrete data such as the integers. The 'Newton' there has to do to with the well known numerical method due to Isaac Newton for approximating the zeros of continuous functions under suitable conditions that is taken to be a paradigm of scientific computing. Finally, the meaning of "Turing meets Newton" in the title of Blum's article has another, more particular aspect: in connection with the problem of controlling errors in the numerical solution of linear equations and inversion of matrices,Turing (1948) defined a notion of *condition* for the relation of changes in the output of a computation due to small changes in the input that is analogous to Newton's definition of the derivative.

The thrust of Blum's 2004 article was that the BSS model of computation on the reals is the appropriate foundation for scientific computing in general. By way of response, two years later another very interesting and equally readable article appeared in the *Notices*, this time by Mark Braverman and Stephen Cook (2006) entitled "Computing over the reals: Foundations for scientific computing," in which the authors argued that the requisite foundation is provided by a quite different "bit computation" model, that is in fact *prima facie* incompatible with the BSS model. The bit computation model goes back to ideas due to Stefan Banach and Stanislaw Mazur in the latter part of the 1930s, but the first publication was not made until Mazur (1963). In the meantime, the model was refined and improved by Andrzej Grzegorczyk (1955) and independently by Daniel Lacombe (1955) in terms of a theory of recursively computable functionals of the sort familiar to logicians. Terminologically, something like "effective approximation

computability" is preferable to "bit computability" as a name for this approach in its applications to analysis.[1]

There are functions that are computable in each of these two models of computation over the reals that are not computable in the other. For example, the exponential function is computable in the effective approximation model but not in the BSS model. And in the latter—but not in the former—given a non-trivial polynomial p over the rationals, one can compute the function of x that is 1 just in case $p(x) = 0$ and is 0 otherwise. *Despite their incompatibility, is there any way that these can both be considered to be reasonable candidates for computation on the real numbers?* As we shall see, an obvious answer is provided very quickly by the observation that the BSS model may be considered to be given in terms of computation over the reals as an *algebraic structure*, while that of the effective approximation model can be given in terms of computation over the reals 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.*

After explaining the BSS and effective computation models respectively in sections 2 and 3 below, my main purpose here is to describe three theories of computation over (more or less) arbitrary structures 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 earlier work of Richard Platek and Yiannis 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.

Before engaging in all that, let us return to the issue of computation over the reals as a foundation for *scientific computation*, aka *computational science* or *numerical analysis*. That subject is problem oriented toward the development of techniques for such diverse tasks as solving systems of one or more linear and polynomial equations, interpolation from data, numerical integration and differentiation, determination of maxima and minima, optimization, and solutions of differential and integral equations. Though nowadays these techniques are formulated as programs to be carried out on

[1] These two are not the only theories of computation over the real numbers. See Weirauch (2000) for a useful survey of various such notions.

actual computers—large and small—many of them predate the use of computers by hundreds of years (*viz*., Newton's method, Lagrange interpolation, Gaussian elimination, Euler's method, etc., etc.). 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 over the reals as exact objects. For, in practice, those computations are made in "floating point arithmetic" using finite decimals with relatively few significant digits, for which computation *per se* simply reduces to computation with rational numbers.

Besides offering a theory of computation on the real numbers, the main emphasis in the articles of Blum (2004), Braverman and Cook (2006), and the book Blum, et al. (1997) is on their relevance to the subject of scientific computation in terms of measures of complexity. These use, among other things, analogues to the P and NP classifications (due to Cook) from the theory of discrete computation.[2] In addition to examples of complexity questions in numerical analysis, they are illustrated with more recent popular examples such as those having to do with the degree of difficulty of deciding membership in the Mandlebrot set or various Julia sets. 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.

2. The BSS model. A brief but informative description is given in Blum (2004), p. 1028, while a detailed exposition is to be found in Blum, et al. (1997), Chs. 2 and 3, with the first of these chapters devoted to what is called the finite dimensional case and the second to its extension to the infinite dimensional case. As stated in these sources, the BSS definition makes sense for any ring or field A, not only the reals $\mathbb{R}$ and the complex

[2] It should be mentioned that pioneering work on a notion of polynomial time complexity in analysis using the effective approximation approach had been made by Ko and Friedman (1982) and Ko (1991); cf. the survey Ko (1998). For further work in this direction, see Stoltenberg-Hansen (1999) and Bauer (2002).

numbers $\mathbb{C}$, and so on the face of it, it is 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 then to branch to further instructions accordingly. (In case A is ordered, one can also test similarly whether $x < y$ or not.) In the BSS description, an algorithmic procedure is given by a directed graph whose top node represents the input stage and passage from a given node to a successor node is made either by a direct computation, which in the case of a ring is effected by a polynomial function (and in the case of a field by a rational function), or by a decision step whose branches may proceed to a new node or return to an earlier node. In the finite-dimensional case, the inputs are finite sequences of any fixed length, while in the infinite-dimensional case they are sequences of arbitrary length from A. The finite dimensional case is illustrated by the Newton algorithm for $\mathbb{R}$ or $\mathbb{C}$ as follows: given a rational function f, to find a zero of f within a prescribed degree of accuracy $\varepsilon > 0$, one starts with an input x, and successively updates that by replacing x by $x - f(x)/f'(x)$, until one reaches a value of x (if at all) for which $|f(x)| < \varepsilon$. The infinite dimensional case is illustrated by a BSS algorithm to decide whether or not m given polynomials $f_1, \ldots, f_m$ in n variables over $\mathbb{C}$ have a common zero, where the procedure is to be described for n, m arbitrary; this is related in an essential way to the Hilbert Nullstellensatz of 1893.[3]

It is pointed out in Blum, et al. (1997) that a BSS algorithm for the finite-dimensional case may be thought of more concretely in terms of register machines as a

[3] As stated by Blum (2004), p. 1027, Hilbert's Nullstellensatz (theorem on the zeros) asserts in the case of the complex numbers $\mathbb{C}$ that if $f_1,\ldots,f_m$ are polynomials in $\mathbb{C}[\underline{x}]$ for indeterminates $\underline{x} = (x_1,\ldots,x_n)$, then $f_1,\ldots,f_m$ have *no* common zero in $\mathbb{C}^n$ iff there are polynomials $g_1,\ldots g_m$ in $\mathbb{C}[\underline{x}]$ such that $\sum g_i f_i = 1$. This gives a semidecision procedure by searching for such g_i. That is turned into a BSS algorithm to decide whether or not the f_i have a common zero by use of effective bounds on the degrees of possible such g_i found by Grete Hermann in 1926.

direct generalization of the notion due to Shepherdson and Sturgis (1963).[4] Actually, such a generalization to arbitrary fields was already made by Herman and Isard (1970). For the infinite dimensional case, the BSS machine picture is treated instead as a form of a Turing machine with a two-way infinite tape whose squares may be occupied by elements of the ring or field A; the infinite tape allows for inputs consisting of sequences of arbitrary length. Alternatively, it could be treated as a register machine with so-called stack registers. Indeed, this falls under a very general adaptation of the register machine approach to arbitrary structures that was made by Friedman (1971); this will be described in sec. 4 below.

It may be seen that in the case of rings, resp. fields, only piecewise polynomial functions, resp. rational functions, are BSS computable. In particular, the Newton algorithm can only be applied to such. But one could add any continuous function f to the basic field structure and relativize the notion of BSS computable function to that. Of course, doing so affects the complexity questions that are of major concern in Blum, et al. (1997). In the opposite direction, one may ask what BSS algorithms can actually be carried out on a computer. Tarski's decision procedure (1951) for the algebra of real numbers may be considered as a special case of such. Its centerpiece method reduces the question of whether or not a system of polynomial equations and inequalities in one variable with coefficients in $\mathbb{R}$ has a common solution, to a quantifier-free condition on the coefficients of the polynomials. On the face of it, Tarski's procedure runs in time complexity as a tower of exponentials. That was cut down considerably to doubly exponential upper bounds by George Collins in 1973 using a new method that he called Cylindrical Algebraic Decomposition (CAD); Collins' original work and many relevant articles are to be found in the valuable source Caviness and Johnson (1998). The CAD algorithm has actually been implemented for computers by one of Collins' students, and in a modified form in the system *Mathematica*; that works in reasonable time for polynomials of relatively low degree. But in principle, Fischer and Rabin (1974, reprinted in Caviness and Johnson), give an EXPTIME lower bound of the form 2^{cn} for

[4] See Cutland (1980) for a nice exposition and development of recursion theory on the basis of register machines. This is also referred to by some as the Random Access Model (RAM).

deciding for sentences of length n whether or not they are true in $\mathbb{R}$, no matter what algorithm is used; the same applies even with non-deterministic algorithms, such as via proof systems. They also showed that the "cut-point" by which EXPTIME sets in, i.e. the least n_0 such that for all inputs of length $n \geq n_0$, at least 2^{cn} steps are needed, is not larger than the length of the given algorithm or axiom system for proofs. Thus real algebra is definitely infeasible on sufficiently large, though not exceptionally large inputs.

3. The theory of computability over $\mathbb{R}$ by effective approximation. In contrast to the BSS model, the effective approximation or "bit computation" model of computation over the reals is an analytic theory, specific to $\mathbb{R}$, though the same idea can be generalized to complete separable metric spaces relative to an enumerated dense subset. There are actually two approaches to this model of computation, one that works with sequences approximating arguments and values, that is referred to here as S-effective approximation, and one that works by approximation of functions by polynomials, referred to as P-effective approximation. It turns out that these are equivalent.

To show that a real valued function $f: \mathbb{I} \to \mathbb{R}$, where $\mathbb{I}$ is a finite or infinite interval, is computable by S-effective approximation, given any x in $\mathbb{I}$ as argument to f, one must work *not* with x but rather with an arbitrary *sequential representation of* x, i.e. with a Cauchy sequence of rationals $\langle q_n \rangle_{n \in \mathbf{N}}$ which approaches x as its limit, in order to effectively determine another such sequence $\langle r_m \rangle_{m \in \mathbf{N}}$ which approaches f(x) as limit.[5] The sequences in question are functions from $\mathbb{N}$ to $\mathbb{Q}$, and so the passage from $\langle q_n \rangle_{n \in \mathbf{N}}$ to $\langle r_m \rangle_{m \in \mathbf{N}}$ is given by a computable type-2 functional F. One can standardize the requirements here by restriction, for example, to dyadic rationals $q_n = \varphi(n)/2^n$ where $\varphi: \mathbb{N} \to \mathbb{Z}$ is such that

(1) $$|x - \varphi(n)/2^n| \leq 1/2^n \text{ for all } n.$$

[5] There is no way to work effectively with real numbers given as Dedekind sections in the rational numbers.

In these terms, the requirement for S-effective approximation computability of f reduces to obtaining a computable functional F: $\mathbb{Z}^{\mathbb{N}} \to \mathbb{Z}^{\mathbb{N}}$ such that for any φ: $\mathbb{N} \to \mathbb{Z}$ satisfying (1) and for $F(\varphi) = \psi$ we have

(2) $\qquad (\forall n)\, |x - \varphi(n)/2^n| \leq 1/2^n \Rightarrow (\forall m)\, |f(x) - \psi(m)/2^m| \leq 1/2^m.$

When this holds for all x in the domain of f we say that F effectively represents f. This notion of computability of real functions is due independently to Grzegorczyk (1955) and Lacombe (1955). It is illustrated in Braverman and Cook (2006) by proof of the bit-computability of $f(x) = e^x$ using its usual power series expansion, thus showing that for many such functions, one goes well beyond BSS computability over the reals in this model.

Using the effective correspondence of $\mathbb{Z}$ with $\mathbb{N}$, the preceding explanation of S-effective approximation computability of real functions reduces to the explanation from classical recursion theory of the notion of effectively computable functional F: $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$. Write $\mathcal{T}$ for the class $\mathbb{N}^{\mathbb{N}}$ of all *total* functions from $\mathbb{N}$ to $\mathbb{N}$, and $\mathcal{P}$ for the class of all *partial* functions from $\mathbb{N}$ to $\mathbb{N}$. The original explanation of what is an effectively computable functional (of type 2 over N) was that given by Kleene (1952) p. 326 for F: $\mathcal{P} \to \mathcal{P}$, in terms of Herbrand-Gödel calculability from a suitable system of equations E with a symbol for an arbitrary partial function φ and a symbol ψ for a function defined by E from φ. The idea is that for any φ in $\mathcal{P}$, the values $\psi(m)$ can be calculated by the equational rules from E together with the diagram (i.e. formal graph) of φ. It is immediate that any partial recursive F is monotonic in its arguments, i.e. if φ and φ' are partial functions with φ' an extension of φ then $F(\varphi')$ is an extension of $F(\varphi)$. Moreover we have the continuity property that if $F(\varphi) = \psi$ then each value $\psi(n)$ depends only on a finite number of values of φ, i.e. there is a finite subfunction φ_0 of φ such that if φ' is any extension of φ_0 and $\psi' = F(\varphi')$ then $\psi'(m) = \psi(m)$. Combining these facts, one has that $F(\varphi)$ is the union of $F(\varphi_0)$ for all finite $\varphi_0 \subseteq \varphi$. Kleene's main result for the partial recursive functionals is his ("second") Recursion Theorem, according to which each such F has a least fixed point, obtained by taking the union of all finite iterations of F starting with the empty function. An alternative approach to partial recursive functionals (that

yields the same class as Kleene's) is via recursive operators, as exposited for example in Cutland (1980), Ch. 10; it only presumes the notion of partial recursive function, independently of how that is defined.

Now the computable functionals F: $\mathcal{T}\rightarrow\mathcal{T}$ may be defined to be those partial recursive functionals F whose value for each total function φ is a total function F(φ). (It is not enough to restrict a partial recursive functional to $\mathcal{T}$, since its value on a total argument f may still be partial.) But there are several other ways of defining which are the computable functionals F: $\mathcal{T}\rightarrow\mathcal{T}$ without appealing to the notion of partial recursive functional. One is due to Grzegorczyk (1955) by means of a direct generalization of Kleene's schemata for the general recursive functions using both primitive recursion and the least number operator μ. Grzegorczyk deals with F(φ)(x), where φ = $\varphi_1,\ldots,\varphi_j$ is a sequence (possibly empty) of total functions, and x is a sequence $x_1,\ldots,x_k$ of numerical arguments with $k \geq 1$. For simplicity, we shall write F(φ, x) for Gregorczyk's F(φ)(x). These functionals reduce to functions of natural numbers F(x) when j = 0. A basic computable functional F taken in the schemata is that for application, i.e. F(φ, x) = φ(x). Then in addition to the primitive recursive schemata[6] relativized uniformly to function arguments φ, one has a scheme for the μ (minimum) operator formulated as follows. Suppose given computable F(φ, x, y) such that $\forall\varphi, x\, \exists y\, [F(\varphi, x, y) = 0]$; then the functional G defined by

$$G(\varphi, x) = \mu y\, [F(\varphi, x, y) = 0]$$

is computable. An equivalent definition of computable functionals from total functions to total functions has been given by Weirauch (2000) via uniform oracle Turing machines, called by him the Type-2 Theory of Effectivity (TTE). In my view, Kleene's notion of partial recursive functional is the fundamental one, in that it specializes to Grzegorczyk's (or Weirauch's) in the following sense: if F is a partial recursive functional F: $\mathcal{P}\rightarrow\mathcal{P}$, 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.[7]

---

[6] Actually, Grzegorczyk only assumes special consequences of primitive recursion from which the general primitive recursive schemata are inferred.

[7] One proof is as follows. Let E be a system of equations used to define F, and consider ψ = F(φ) for any total φ; by assumption, for each i there is a unique j and a derivation of

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 $\mathbb{I}$. In particular, take $\mathbb{I}$ to contain 0 and consider the function f on $\mathbb{I}$ that is 0 on x just in case $x < 0$ or $x > 0$ and otherwise is 1; since f is not continuous on $\mathbb{I}$ it is not computable in the S-approximation sense. Thus, unlike the BSS model, the order relation on $\mathbb{R}$ is not computable. In general, in the case that $\mathbb{I}$ is a finite closed interval, if f is computable on $\mathbb{I}$ in the S-effective approximation sense, one has that it is effectively uniformly continuous relative to the sequential limit representations of its arguments (Grzegorczyk 1955 p.192). As we shall see, this connects with the Bishop constructive approach to continuous functions, to be discussed in the final section.

Let us turn now to the notion of P-effective approximation computability due to Marian B. Pour-El (1974). That is suggested by the Weierstrass Approximation Theorem, according to which every continuous function on a closed interval $\mathbb{I}$ is uniformly approximable by a sequence of polynomials over $\mathbb{Q}$. There is an effective enumeration of all possible polynomials $P_n(x)$ over $\mathbb{Q}$ with non-zero rational coefficients. Then Pour-El defines f with domain $\mathbb{I}$ to be computable by P-effective approximation if there are recursive functions φ and ψ such that for all $M > 0$, all $n \geq \psi(M)$, and all x in $\mathbb{I}$,

(6) $$|f(x) - P_{\varphi(n)}(x)| \leq 1/2^M .$$

She extends this to f defined on $\mathbb{R}$, by asking for a recursive function φ of two variables such that for all k, and all M, n as above and all x in [-k, k],

(7) $$|f(x) - P_{\varphi(k,n)}(x)| \leq 1/2^M .$$

It is proved in Pour-El and Caldwell (1975) that P-effective computability is equivalent to

---

$\psi(i) = j$ from E together with the diagram of φ. Any such derivation makes use only of a finite subfunction $\varphi|k$ of φ. Now let $D(\varphi, i, j, k, d) = 0$ hold when d is a code of a derivation from $\varphi|k$ of $\psi(i) = j$, otherwise = 1. D is primitive recursive uniformly in φ, and for each φ and i there are j, k, d such that $D(\varphi, i, j, k, d) = 0$. Finally, $F(\varphi, i)$ is the first term of the least triple $\langle j, k, d \rangle$ such that $D(\varphi, i, j, k, d) = 0$.

S-effective computability of functions on any closed interval $\mathbb{I}$ and on the full real line $\mathbb{R}$. Slightly more general results with an alternative proof are given in Shepherdson (1976). Though the notions of P-effective computability are simpler than those of S-effective computability, Shepherdson remarks that insofar as actual computation is concerned, "the values are still given via approximating sequences [of rationals to reals]; this is inescapable."

4. The view from generalized recursion theory (g. r. t.); two theories of computability on arbitrary structures. Beginning in the 1960s and continuing through the 1970s and beyond there was much work on generalizations of recursion theory to arbitrary structures. One of these generalizations was made by Harvey Friedman (1971) by adaptation of the register machine approach as explained next.[8] As will be seen below, this approach to computability over arbitrary structures considerably antedates and comprehends the BSS notions.

By a (first-order) structure or algebra $\mathfrak{A}$ is meant one of the form

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

where A is a non-empty set, each c_i is a member of A, each f_i is a partial function of one or more arguments from A to A, and each R_i is a relation of one or more arguments on A.[9] For non-triviality, both k and m are not zero. The signature of $\mathfrak{A}$ is given by the triple (j, k, m) and the arity of each f_i and each R_i. Of special note is that the test for equality of elements of A is *not* assumed as one of the basic operations; rather, if equality is to be a basic test, that is to be included as one of the relations R_i. A *finite algorithmic procedure* (fap) π on $\mathfrak{A}$ is given by a finite list of instructions $I_1, I_2, \ldots, I_t$ for some t, with I_1 being the initial instruction and I_t the terminal one. The machine has register names $r_0, r_1, r_2, \ldots$, though only a finite number are needed for any given computation, namely those mentioned in π; the register r_0 is reserved for the output. The r_i may also be thought of as variables. The fap π may be used to calculate a partial n-ary function f on A^n to A for any

[8] Friedman also considered a variant approach using infinite tapes as in Turing machines and as in Blum (2004).
[9] Friedman allowed partial relations in a suitable sense.

n. Given an input $(x_1, \ldots, x_n)$, one enters x_i into register r_i, and proceeds to I_1. Each instruction other than I_t has one of the following four forms:

(2) $r_a := r_b$

$r_a := c_i$

$r_a := f_i(r_{b1}, \ldots, r_{bj})$, for j-ary f_i

if $R_i(r_{b1}, \ldots, r_{bj})$ then go to I_u else I_v, for j-ary R_i.

In the first three cases, one goes to the next instruction after executing it. The computation terminates only if the third instruction is defined at each stage where it is called and if one eventually lands in I_t, at which point the contents of register r_0 is the value of $f(x_1, \ldots, x_n)$. An n-ary relation R is decidable by a fap π if its characteristic function is computable by π. The class of fap computable partial functions on $\mathfrak{A}$ is denoted by **FAP**($\mathfrak{A}$).

For the structure $\mathfrak{N} = (\mathbb{N}, 0, Sc, Pd, =)$, where $\mathbb{N}$ is the set of natural numbers and Sc and Pd are respectively the successor and predecessor operations (taking Pd(0) = 0), **FAP**($\mathfrak{N}$) is equal to the class of partial recursive functions. For general structures $\mathfrak{A}$, Friedman (1971) also introduced the notion of *finite algorithmic procedure with counting*, in which certain registers are reserved for natural numbers and one can perform the operations and tests on the contents of those registers that go with the structure $\mathfrak{N}$. Then **FAPC**($\mathfrak{A}$) is used to denote the partial functions on A computable by means of such procedures.

The notion of finite algorithmic procedure is directly generalized to many-sorted structures

(3) $\mathfrak{A} = (A_1, \ldots A_n, c_1, \ldots, c_j, f_1, \ldots, f_k, R_1, \ldots, R_m)$,

with the arity modified accordingly, while each register comes with a sort index limiting which elements can be admitted as its contents. In particular, **FAPC**($\mathfrak{A}$) can be identified with **FAP**($\mathfrak{A}$, $\mathfrak{N}$) where ($\mathfrak{A}$, $\mathfrak{N}$) denotes the structure $\mathfrak{A}$ augmented by that for $\mathfrak{N}$. A further extension of Friedman's notions was made by Moldestad et al. (1980a, 1980b), with *stack registers* which may contain finite sequences of elements of any one of the basic domains A_i, including the empty sequence. The basic operations for such a register are *pop* (remove the top element of a stack) and *push* (add the contents of one of the registers of type A_i). This leads to the notion of what is computable by *finite algorithmic*

procedures with stacks, **FAPS**($\mathfrak{A}$), where we take the structure $\mathfrak{A}$ to contain with each domain A_i the domain A_i^* of all finite sequences of elements of A_i, and with operations corresponding to pop and push. If we want to be able to calculate the length n of a stack and the qth element of a stack, we need also to have the structure $\mathfrak{N}$ included. This leads to the notion of *finite algorithmic procedure with stacks and counting,* whose computable partial functions are denoted by **FAPCS**($\mathfrak{A}$). In the case of the structure $\mathfrak{N}$, by any one of the usual primitive recursive codings of finite sequences of natural numbers, we have

(4) $$\mathbf{FAP}(\mathfrak{N}) = \mathbf{FAPC}(\mathfrak{N}) = \mathbf{FAPS}(\mathfrak{N}) = \mathbf{FAPCS}(\mathfrak{N}).$$

Trivially, in general for any structure $\mathfrak{A}$ we have the inclusions,

(5) $$\mathbf{FAP}(\mathfrak{A}) \subseteq \mathbf{FAPC}(\mathfrak{A}) \subseteq \mathbf{FAPCS}(\mathfrak{A}), \text{ and}$$

$$\mathbf{FAP}(\mathfrak{A}) \subseteq \mathbf{FAPS}(\mathfrak{A}) \subseteq \mathbf{FAPCS}(\mathfrak{A}).$$

It is proved in Moldestad et al. (1980b) that for each of these inclusions there is a structure $\mathfrak{A}$ which makes that inclusion strict.

Consider the structure of real numbers as an ordered field,

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

It is stated in Friedman and Mansfield (1992), p. 298, that "with a little programming" the **FAP**($\mathfrak{R}$) functions are exactly the same as the BSS functions computable in the finite dimensional case, and the **FAPS**($\mathfrak{R}$) functions are exactly the same as the BSS functions computable in the infinite dimensional case. It also follows from Friedman and Mansfield (1992), p. 300, that **FAPS**($\mathfrak{R}$) = **FAPCS**($\mathfrak{R}$), because $\mathfrak{N}$ can be embedded in $\mathfrak{R}$. Appropriate generalizations of these results hold for arbitrary rings and fields $\mathfrak{A}$, ordered or unordered.

An alternative approach to computability over arbitrary structures due to John Tucker and Jeffery Zucker that they developed over a number of years is provided in their very usefully detailed expository piece, Tucker and Zucker (2000); this uses definition by schemata or procedural statements rather than machines. Their approach works over many-sorted algebras

(6) $$\mathfrak{A} = (A_1,\ldots A_n, c_1,\ldots,c_j, f_1,\ldots,f_k),$$

where the f_i may be partial. By a *standard structure* $\mathfrak{A}$ is one that includes the structure $\mathfrak{B}$ with domain{t, f} and basic Boolean functions as its operations. Then relations are

treated as (possibly partial) functions into {t, f}, and branching on a relation is executed via the *if…then…else* command. The basic notion of computability for standard algebras is given by *"while" schemata* generated by the following rules (op. cit. p. 362) for procedure statements S:

(7) $S :: = skip|x: = t|S_1;S_2|$if b then S_1 else $S_2|$while b do S,

where 'b' is a Boolean term. The set of partial functions computable on $\mathfrak{A}$ by means of these rules or schemata is denoted by **While**($\mathfrak{A}$). Then to deal with computability with counting, Tucker and Zucker simply expand the algebra $\mathfrak{A}$ to the algebra ($\mathfrak{A}$, $\mathfrak{N}$). To incorporate finite sequences for each domain A_i, they make a *further* expansion of that to $\mathfrak{A}$*. This leads to the following notions of computability over A: **While**$^{\mathfrak{N}}$($\mathfrak{A}$) and **While***($\mathfrak{A}$), given simply by **While**($\mathfrak{A}$, $\mathfrak{N}$) and **While**($\mathfrak{A}$*), respectively. The following result is stated in Tucker and Zucker (2000) p. 487 for any standard algebra $\mathfrak{A}$:

(8) **While**($\mathfrak{A}$) = **FAP**($\mathfrak{A}$), **While**$^{\mathfrak{N}}$($\mathfrak{A}$) = **FAPC**($\mathfrak{A}$), and

While*($\mathfrak{A}$) = **FAPCS**($\mathfrak{A}$).

In other words, we have a certain robustness of computability in the **While*** sense. Tucker and Zucker (1988) present a number of arguments in favor of a generalized Church-Turing thesis for computability, according to which the functions that effectively computable on a many-sorted algebra $\mathfrak{A}$ in the informal sense are precisely the functions that are **While*** computable on $\mathfrak{A}$; see also Tucker and Zucker (2000) pp. 493ff for a briefer sketch of these arguments.

So far, we are still dealing with essentially algebraic notions of computability. To extend the **While** approaches to notions of computability on the reals, it would seem that topological considerations must be brought to bear. That is done in a more general setting in Sec. 7 (pp. 451-478) of Tucker and Zucker (2000). It is more difficult to describe informally the several ways that is carried out loc. cit., so I shall simply give the names of the notions introduced there, with the hope that they at least indicate what is involved. First of all, the structures $\mathfrak{A} = (A,\ldots)$ dealt with are *topological partial algebras*, where the partiality essentially has to do with the Boolean-valued functions of equality and—in the case of $\mathbb{R}$—order; as total functions these are discontinuous, so one must replace them by partial functions which are undefined at (x, y) in A when x = y

and—in the case of $\mathbb{R}$—when neither $x < y$ nor $y < x$. If the other basic functions on A are taken to be continuous then so also are all the **While*** computable functions. That is then specialized to *metric partial algebras* which are used to explain *effectively uniform **While**, resp. **While***,* and then *approximable computability* on $\mathfrak{A}$ and *effective Weierstrass approximable computability* on $\mathfrak{A}$; these three notions are shown to be equivalent for suitable $\mathfrak{A}$ (op. cit., p. 473). The Weierstrass notion is a generalization of Pour-El's P-computability that was described above. When further specialized to functions on $\mathbb{I}$ to $\mathbb{R}$, where $\mathbb{I}$ is a closed interval, they are further shown (op. cit., p. 474) to be equivalent to Grzegorczyk-Lacombe computability, i.e. S-effective computability described above.[10] Thus the notions of **While** and **While*** computability serve to subsume under a single framework the notions of computability in the BSS sense with those computable in the effective approximation sense.

Another framework from g.r.t. that does not require direct appeal to topological notions for the two notions of computability on the real numbers is provided in the next section.

5. The higher type approach to computation on arbitrary structures. From the schematic point of view, computable functions are generated from given functions by explicit definition and by recursion, i.e. definition of a function in terms of itself. Abstractly, recursion is given by a functional equation, $f = F(f)$, which determines a unique function f as the least fixed point (LFP) of F only if F is monotonic; moreover, the LFP of F is in general a partial function. (For reasons that will be seen below, we now use 'f', 'g', … for partial function arguments of functionals in place of 'φ', 'ψ', … as was done in sec. 3 above.) But then one has to ask where F comes from if it is not itself explicitly generated; that would require determining it as the LFP of a higher type operator G, and so on. This idea formed the underpinning of Richard Platek's definition of computability over fairly arbitrary structures $\mathfrak{A}$ in his famous (but regrettably never published) Stanford PhD dissertation, Platek (1966), in terms of a hierarchy of monotonic

[10] For continuations of this work on computation on metric partial algebras, see Tucker and Zucker (2004, 2005).

partial functionals of arbitrary finite type over the domains of $\mathfrak{A}$. That allows one to start not only with given functions over $\mathfrak{A}$ but also given functionals in that hierarchy. For the applications that concern us here, it would be sufficient to start with functionals of type level ≤ 2 over the domains of $\mathfrak{A}$. Platek showed that in that case, everything of type level ≤ 2 that can be generated via recursion in higher types from such initial data can already be generated via explicit definition and LFP definition with F of type level equal to 2.

Yiannis Moschovakis also took the LFP approach restricted to functionals of type level ≤ 2 in his explanation of the notion of algorithm over arbitrary structures in his papers Moschovakis (1984, 1989), featuring simultaneous LFP definitions, though those can be eliminated in favor of single LFP definitions of the above form. Both the Platek and Moschovakis approaches are *extensional* in a sense to be described below. In order to tie that up both with computation over abstract data types and with Bishop's approach to constructive mathematics, in a pair of papers Feferman (1992a, 1992b), I extended the use of LFP schemata to cover *intensional* situations, by requiring each basic domain A_i of $\mathfrak{A}$ to be equipped with an equivalence relation $=_i$ which the functions and functionals must preserve.[11] But unlike the approach to computation over $\mathbb{R}$ taken op. cit. in which real numbers are dealt with as one of the basic domains, I here treat them as genuinely type 1 objects, via functions on $\mathbb{N}$ representing Cauchy sequences of rational numbers. For simplicity, I will reserve description of the intensional approach to the case of computation over $\mathbb{N}$, so that in effect each $=_i$ is taken to be the identity relation; but intensionality has to be revisited for type 1 objects as will be explained below.

In more detail, this is how the development proceeds for what I call *Abstract Computation Procedures* (ACPs) over an algebra

(1) $\mathfrak{A} = (A_0, A_1, \ldots, A_k, F_0, \ldots, F_m)$,

where each A_i is non-empty and the F_js are individuals, functions or functionals of type level 2 over the A_i satisfying a monotonicity condition to be explained below; A_0 is fixed to be the booleans $\{\mathbf{t}, \mathbf{f}\}$. We use letters f, g, h, … to range over partial functions of arbitrary many-sorted arities given by arguments ranging over some finite product, possibly empty, of the A_is with values in some A_j; in case the product is empty, such f is

[11] See also Feferman (1996) where I treated streams in the extensional interpretation.

simply identified with an element of A_j. The arity of f is determined by a pair $\sigma = (\underline{i}, j)$ where $\underline{i} = (i_1,\ldots i_\nu)$ lists the sorts of the product domain. Given f of arity σ, $\underline{x} = (x_1,\ldots,x_\nu)$ of arity $\underline{i}$, and y of sort j, we write $f(\underline{x}) \simeq y$ when $f(\underline{x})\downarrow$ (i.e. $f(\underline{x})$ is defined) and the value of $f(\underline{x})$ is y. Given f, g of arity $(\underline{i}, j)$ we write $f \subseteq g$ if whenever $f(\underline{x}) \simeq y$ then $g(\underline{x}) \simeq y$.

We can now turn to the functionals (which may reduce to functions or individuals), for which we use the letters F, G, H,… . These have both a sequence $\underline{f} = (f_1,\ldots,f_\mu)$ of partial function arguments of arities $\underline{\sigma} = (\sigma_1,\ldots,\sigma_\mu)$, and individual arguments $\underline{x} = (x_1,\ldots x_\nu)$ of arity $\underline{i}$, and have values $F(\underline{f}, \underline{x})$ in a specified domain A_j when defined; the arity of such F is given by the triple $(\underline{\sigma}, \underline{i}, j)$. We allow $\mu = 0$, in which case F reduces to a partial function of arity $(\underline{i}, j)$; we further allow $\nu = 0$ as above, in which case it reduces to an element of A_j when defined. Given $\underline{f}$, $\underline{g}$ of arity $\underline{\sigma}$, write $\underline{f} \subseteq \underline{g}$, if each $f_\xi \subseteq g_\xi$; then F is said to be *monotonic* if whenever $\underline{f} \subseteq \underline{g}$ and $F(\underline{f}, \underline{x}) \simeq y$ then $F(\underline{g}, \underline{x}) \simeq y$. This is automatically the case when there are no function arguments. The basic assumption on the structure $\mathfrak{A}$ above is that each F_k is monotonic for k = 0,…,m. We further assume that the basic boolean functions corresponding to conjunction and negation are among these.

Suppose given monotonic $G(g, \underline{w})$ with a single function argument g of arity $\sigma = (\underline{i}, j)$ where $\underline{w}$ is of arity $\underline{i}$. Then for any g, the function $\lambda\underline{w}.G(g, \underline{w})$ is again of arity σ. Let $\Gamma_G(g) = \lambda\underline{w}.G(g, \underline{w})$, in other words $\Gamma_G = \lambda g\lambda\underline{w}.G(g, \underline{w})$. Then $LFP(\Gamma_G)$ is defined to be the unique function h such that

(2) $\Gamma_G(h) = h$, and if $\Gamma_G(g) = g$ then $h \subseteq g$.

We can now list the schemata for ACPs F, G, H… over A as follows:

I. (Initial functionals) $F(\underline{f}, \underline{x}) \simeq F_l(\underline{f}, \underline{x})$ for $l = 0,\ldots,m$

II. (Identity functions) $F(x) = x$

III. (Application functionals) $F(f, \underline{x}) \simeq f(\underline{x})$

IV. (Conditional definition) $F(\underline{f}, \underline{x}, b) \simeq$[if $b = \mathbf{t}$ then $G(\underline{f}, \underline{x})$ else $H(\underline{f}, \underline{x})$]

V. (Structural) $F(\underline{f}, \underline{x}) \simeq G(\underline{f}_\rho, \underline{x}_\tau)$

VI. (Individual substitution) $F(\underline{f}, \underline{x}) \simeq G(\underline{f}, \underline{x}, H(\underline{f}, \underline{x}))$

VII. (Function substitution) $F(\underline{f}, \underline{x}) \simeq G(\underline{f}, \lambda\underline{u}.H(\underline{f}, \underline{x}, \underline{u}), \underline{x})$

VIII. (Least fixed point) $F(\underline{f}, \underline{x}, \underline{u}) \simeq LFP[\lambda g \lambda \underline{w}.G(\underline{f}, g, \underline{x}, \underline{w})](\underline{u})$.

In IV of this list, 'b' is of boolean sort. In V, ρ: $\{1,\ldots,\mu'\} \rightarrow \{1,\ldots,\mu\}$ for some μ' and $\underline{f}_\rho = (f_{\rho(1)},\ldots f_{\rho(\mu)})$; similarly for τ and $\underline{x}_\tau$. In all the other cases the arities are taken to be the appropriate ones. We denote by **ACP**($\mathfrak{A}$) the set of all F of type levels 0, 1 and 2 generated from the initial $F_0,\ldots,F_m$ specified by $\mathfrak{A}$, and by $\mathbf{ACP}^1(\mathfrak{A})$ ($\mathbf{ACP}^2(\mathfrak{A})$) the subset consisting of the functions of type level 1 (functionals of type level 2) among these.

As is easily seen, the reason the ACPs deserve to be called *abstract procedures* is that they are preserved under isomorphism. That they also deserve to be called *computation procedures*, at least in the case of $\mathfrak{N}$-standard structures $\mathfrak{A}$ with arrays, is due to the result below of Xu and Zucker (2005) below. A structure $\mathfrak{A}$ is $\mathfrak{N}$-standard if it is an expansion of the structure $\mathfrak{N} = (\mathbb{N}, 0, Sc, Pd)$. An $\mathfrak{N}$-standard structure has arrays if with each basic domain A_i is associated the domain of all finite sequences from A_i, with the appropriate operations of length, term, expansion and restriction. Let **ACP**($\mathfrak{A}$) denote the set of functions generated by the schemata I-VIII above. Xu and Tucker proved:

(3) If $\mathfrak{A}$ is an $\mathfrak{N}$-standard structure with arrays, then

$$\mathbf{While^*}(\mathfrak{A}) = \mathbf{ACP}^1(\mathfrak{A}).$$

We also have a matchup with the Moschovakis (1984) theory of algorithms by the result of Feferman (1992b) sec.9 that the ACPs are closed under simultaneous LFP recursion.[12]

Since it was seen in the preceding section that the BSS machine model of computation on the real numbers (and on algebraic structures more generally) is subsumed under **While*** computability, to show that we have a generalization of both that and the effective approximation approach to computation on the reals, we specialize to the case of ACPs over the structure $\mathfrak{N}$, which by the usual coding, includes the associated structure with arrays. And for this, one simply comes down to showing that

(4) $\mathbf{ACP}^2(\mathfrak{N})$ = the partial recursive functionals over the natural numbers.

For it is easy to show that every partial recursive function is generated by the ACPs over $\mathfrak{N}$, from which one obtains all the partial recursive functionals in the guise of recursive

[12] See also Feferman (1996), Appendix A. Note also that Appendix C of that paper contains several corrections to Feferman (1992b).

operators as mentioned above. To prove the converse, one shows inductively that every F in **ACP**($\mathfrak{N}$) is a partial recursive function if of type level 1, and a partial recursive functional if of type level 2. The crucial step is to show that the if G is a partial recursive functional then the function LFP(Γ_G) is a partial recursive function; for details, see Feferman (1992b), secs. 10.2 and 11. So now the S-approximation theory of effective computability of functions of real numbers is explained essentially as in sec. 3 above in terms of total recursive functionals in $\mathbf{ACP}^2(\mathfrak{N})$. That is, Cauchy sequences of rational numbers with effective moduli of convergence are represented in one way or another by a class Rep of total functions on $\mathbb{N}$. For any two such functions, $f \equiv g$ is defined to hold if the corresponding Cauchy sequences have the same limit, and a function F of real numbers is represented by a functional F if for each real number x and each f representing x we have that F(f) is a function representing F(x). Finally, a functional F serves to do this if it maps Rep to Rep and if $f \equiv g$ implies $F(f) \equiv F(g)$.

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 relevant questions of complexity.

Coming back to basics, the foregoing illustrates another fundamental issue, namely 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 that has been glossed over in the above explanation of ACPs. Namely, functions f, g, h,… there are tacitly understood in the usual set-theoretic sense for which the extensionality principle—that extensional equality implies identity—holds, i.e. if $f(n) = g(n)$ for all n in $\mathbb{N}$, then $f = g$. By the *intensional recursion-theoretic interpretation* of **ACP**($\mathfrak{N}$) I mean what one gets by taking the function variables f, g, h, … to range instead over *indices* of partial recursive functions (of the appropriate arities), rather than the functions themselves. To connect this with the ordinary recursion theoretic interpretation, let us write $\{f\}(\underline{x})$ for $f(\underline{x})$ in the above when f is such an index. Then $f \subseteq g$ and monotonicity of functionals is defined as above; write $f \equiv g$ for $f \subseteq g$ and $g \subseteq f$, i.e. if

f and g are extensionally equal. Now one proves inductively for this interpretation that each F in $\mathbf{ACP}^2(\mathfrak{N})$ preserves extensional equality and hence is an effective operator in the sense of Myhill and Shepherdson (1955), i.e. if $f \equiv g$ then $F(f) \equiv F(g)$. That is also used to show that we have closure in this interpretation under the LFP scheme, since by the Myhill-Shepherdson theorem, every effective operator is the restriction to the partial recursive functions of a partial recursive functional; for more details, see Feferman (1992b), secs. 10.4 and 11. 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.

6. Explicit mathematics and the Bishop approach to constructive analysis. In 1967 Errett Bishop published his ground-breaking book, *Foundations of Constructive Analysis.* Bishop had for many years been a practicing analyst in the classical tradition to which he contributed important work on Banach spaces, operator algebras, function algebras and the theory of functions of several complex variables. In the mid-60s, while spending a year at the Miller Institute in U. C. Berkeley, Bishop had a radical change of mind about how mathematics ought to be developed. Namely he became convinced that it should be carried out constructively so that each theorem has "numerical meaning", i.e. can in principle have a computational interpretation. But he found that the most sustained previous effort to redevelop mathematics constructively—in the work of L. E. J. Brouwer and his school of intuitionism—was very unsatisfactory "partly by extraneous peculiarities of Brouwer's system which made it vague and even ridiculous to practising mathematicians, but chiefly by the failure of Brouwer and his followers to convince the mathematical public that abandonment of the idealistic [i.e., classical] viewpoint would not sterilize or cripple the development of mathematics." (Bishop (1967) p. 2) In its place, Bishop explained a way of developing analysis constructively that could be readily understood by classical mathematicians and yet would be constructively meaningful at the same time. Where Brouwer depended in part on the rejection of classical logical reasoning as exemplified by use of the law of excluded middle to lead to existential conclusions for which one may not have a witness, Bishop depends in part on the systematic replacement of classical notions by related ones in which all witnessing

information is explicitly stated and carried along in proofs. Furthermore, Brouwer depended in his redevelopment of analysis on the use of the intuitive notion of "choice sequence", which has no direct classical interpretation, and on principles concerning that notion which allowed him to prove such classically false statements as that every function on a closed interval of the real numbers is uniformly continuous. Bishop, by contrast, when dealing with functions on a closed interval [a, b], simply restricts himself to those f which are not only uniformly continuous, but carry with them a uniform modulus of convergence function, i.e. an effective function m: $\mathbb{Q} \rightarrow \mathbb{Q}$ such that for each rational $\varepsilon > 0$ we have $m(\varepsilon) > 0$ and for all x, y in [a, b], if $|x - y| < m(\varepsilon)$ then $|f(x) - f(y)| < \varepsilon$. For Bishop, in effect, the objects with which one works are pairs (f, m) of functions satisfying this condition on given closed interval [a, b]. Every theorem in Bishop is also classically true; in practice it gives a constructive substitute for a classical theorem which is equivalent to the latter under the assumption of the law of excluded middle.[13]

With such modifications, Bishop (1967) showed how substantial tracts of modern analysis could be developed in an informal style meeting everyday standards of rigor. Subsequently—in work with one of his students, Henry Cheng—Bishop and Cheng (1972) published an improved version of his theory of measure; a couple of years later, work on constructive probability theory was carried out by another student, Y.-K. Chan (1974). But despite the evidence of its success in carrying out a great deal of mathematics constructively without using strange notions or assumptions, Bishop's approach did not have any major impact, though it did take hold in a small but steadily widening group. One of the first mathematicians outside of his immediate circle who was to take it up was Douglas S. Bridges, who made several contributions to constructive functional analysis in the mid 1970s leading up to the monograph Bridges (1979). Bishop then began a collaboration with Bridges to make substantial additions and improvements in his book that resulted in the volume, *Constructive Analysis,* Bishop and Bridges (1985); that appeared after Bishop's life and career was brought to a premature close by his death due to cancer. There have been a number of further developments in

[13] According to Bishop, only the particular consequences of the law of excluded middle according to which for a function f: $\mathbb{N} \rightarrow \mathbb{N}$ either $(\forall n \in \mathbb{N})\ f(n) = 0$ or $(\exists n \in \mathbb{N})\ f(n) \neq 0$, (that he calls the Limited Principle of Omniscience) are needed for these equivalences.

his school, not only in analysis but also in algebra and topology.[14] But since my main concern here is to relate the essentials of Bishop's approach to contructive mathematics (BCM) to the questions of computation of functions of real numbers, there is no need to refer to anything beyond Bishop and Bridges (1985), abbreviated 'BB' in the following.

Ch. 1 of BB is devoted to Bishop's "Constructivist manifesto"; the mathematical work begins in Ch. 2, on the real numbers and calculus. That begins with some general notions about sets and functions. Membership of an element x in a set A, $x \in A$, is determined by x's meeting certain construction requirements specified by A; these vary from set to set. In addition, each set A carries with it an equivalence relation on A, written $=_A$, which is called the notion of equality for A. If one changes that notion for given construction requirements, one changes the set. In the case of the set $\mathbb{Z}$ of integers (or $\mathbb{Z}^+$ of positive integers), $=_{\mathbb{Z}}$ is taken to be the identity. The set $\mathbb{Q}$ of rational numbers is taken in BB to consist of all pairs (m, n) of integers for which $n \in \mathbf{Z}^+$ and m, n are relatively prime; for this, too, the equality $=_{\mathbb{Q}}$ is the identity. Alternatively, one could take it to consist of all pairs (m, n) of integers for which $n \neq 0$, endowing that set as usual with the equality relation $(m, n) =_{\mathbb{Q}} (m', n')$ iff $m \times n' = m' \times n$.

By an *operation* from a set A into a set B is meant "a finite routine f which assigns an element f(a) of B to each given element a of A. This routine must afford an explicit, finite, mechanical reduction of the procedure for constructing f(a) to the procedure for constructing a." (BB, p.15) By a *function* or *mapping* of a set A into a set B is meant an operation f from A into B such that we have $f(a) =_B f(a')$ whenever $a =_A a'$. The equality of functions f, g from A to B is defined by the condition that for all $a \in A$, $f(a) =_B g(a)$. Note well that this is not the identity relation, since f, g may be given by distinct procedures for computing the same value up to equality in B. By a *sequence* x of elements of a set A is meant a function from $\mathbb{Z}^+$ (or sometimes from $\mathbb{N}$) into A; the nth term of x is given by $x_n = x(n)$; x is also denoted (x_n). Moving on to the real numbers (BB p.18), a sequence $x = (x_n)$ of rational numbers is called *regular* if for all $m, n \in \mathbf{Z}^+$,

[14] See, for example, Mines, Richman and Ruitenberg (1988) for algebra and Spitters (2003) and Bridges and Vita (2006) for analysis.

$|x_m - x_n| \leq 1/m + 1/n$. Thus regular sequences are Cauchy (or fundamental) in the usual sense of the word. The set $\mathbb{R}$ of all real numbers is defined to consist of all regular sequences, with the equality $x =_{\mathbf{R}} y$ defined by: $|x_n - y_n| \leq 2/n$ for all n in $\mathbb{Z}^+$. This is evidently a variant of the explanation of the explicit presentation of real numbers by Cauchy sequences described in sec. 3 above, though with much slower rates of convergence.

The operations of addition, subtraction, multiplication and absolute value are defined in a simple way from regular sequences so as to yield the expected functions on $\mathbb{R}$. For example, x + y is defined to be the sequence z with $z_n = x_{2n} + y_{2n}$ for all n. It is only when we come to inverse for non-zero real numbers that we need to introduce some new witnessing information. Classically, if a Cauchy sequence x converges to a positive real number, there will exist positive integers j and m such that for all $n \geq j$ we have $x_n \geq 1/m$. But constructively there is in general no way to compute effectively such j and m from the given computation procedure used to exhibit x. Bishop's definition for regular sequences x is simply that x *is positive* if for some k, $x_k > 1/k$; given such k, we may choose $m^{-1} \leq (x_k - k^{-1})/2$ and j = m to satisfy the preceding conditions. Thus to exhibit x as a positive real number, we must add the number k as part of its presentation. On the other hand, x is defined to be *non-negative* if $x_n \geq -1/n$ for all positive integers n. It is not constructively true that if x is non-negative then either $x =_{\mathbb{R}} 0$ or x is positive, since we don't have enough information to decide which disjunct holds or to produce a suitable k witnessing the positivity of x if x is not zero. Now, given these notions, we may define $y < x$ to hold if x – y is positive, and $y \leq x$ if x – y is non-negative. And with that, we may proceed to define what is meant by a continuous function f on a closed interval, witnessed by a uniform modulus of continuity function $m(\varepsilon)$ as defined above. Functions f on other intervals are described via the modulus information $m_{a,b}$ associated with each closed subinterval [a, b].

The foregoing should give an idea of how Bishop and his followers proceed to produce constructive substitutes for various classical notions in order to provide a thorough-going constructive redevelopment of analysis. What is not clear from Bishop's

1967 presentation or that of Bishop and Bridges (1985) 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, that may be accomplished by formalizing the work of BB (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 Feferman (1979), p. 173 and pp. 192-197 (cf. also Beeson 1985). Roughly speaking, those account for the computational content of BCM 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 intuitionistic. For the first kind of system, the method of extraction is by one form or another of 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 Feferman (1975) and applied to BCM in Feferman (1979) is of the second kind, and it is only that whose direct interpretation relates it to the theories of computation discussed above. Namely, T_0 has variables of two kinds: individuals a, b, c,…, x, y, z and classes (or "classifications") A, B, C,…, X, Y, Z; the ontological axiom tells us that every class is an individual; the informal meaning is that classes are considered intensionally via their explicit defining properties.[15] The basic relation between individuals, besides that of identity, is a three-placed relation App(x, y, z), also written $xy \simeq z$, satisfying, with suitable constant symbols k, s, p, p_0 and p_1, the conditions for a partial combinatory algebra with pairing and projection operations. The informal meaning of $xy \simeq z$ is that x represents (or codes) a partial operation whose value at y equals z. Thus operations may apply to operations, but also to classes via the ontological axiom. T_0 has a straightforward model in which the individual variables range over the natural numbers and the relation App(x, y, z) holds just in case $\{x\}(y) \simeq z$, in the notation of ordinary recursion theory. On the other hand, within T_0, Bishop's

[15] Later reformulations of systems of explicit mathematics use a relation R(x, X) to express that the individual x represents (or codes) the class X.

constructive analysis is formalized directly following the kinds of explanations sketched above for operations f and sets A (considered as classes with equality relations), functions, $\mathbb{Z}$, $\mathbb{Q}$, regular sequences, $\mathbb{R}$, $=_{\mathbb{R}}$, $<$ and $\leq$ for $\mathbb{R}$, and functions of real variables on closed intervals. Then one can see that the recursion theoretic model of T_0 just described fits the computational content of Bishop's constructive mathematics with the intensional recursion theoretic interpretation of **ACP**($\mathfrak{N}$) described at the end of sec. 5.

As it turns out, and as explained in Feferman (1979), case studies of typical arguments in Bishop's constructive analysis show that it can be formalized in a subsystem of T_0 of the same strength as the system PA of Peano Arithmetic. Further work of Feng Ye (2000) on the constructive theory of unbounded linear operators suggests that in fact a subsystem of the same strength as the system PRA of Primitive Recursive Arithmetic already suffices for that purpose. It is possible that one can push this further by formalization in systems of feasible analysis such as that in Ferreira (1994); that will take much more work.[16] But the 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.[17] 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. [18]

Stanford University
Email: feferman@stanford.edu

References

[16] In a personal communication, Stephen Cook has commented on Ferreira (1994) that "it has a first-order part based on polynomial time functions over a discrete space, but this is supplemented by powerful axioms such as the weak pigeonhole principle which allow existence proofs with no feasible algorithmic content."
[17] See the suggestions that I made in that direction at the conclusion of Feferman (1984).
[18] I would like to thank Michael Beeson, Lenore Blum, Douglas Bridges, Stephen Cook, Jeffery Zucker, and especially the referee for their helpful comments on a draft of this article.

Bauer, A. (2002), A relation between equilogical spaces and type two effectivity, *Mathematical Logic Quarterly* 48, 1-15.

Beeson, M. J. (1985), *Foundations of Constructive Mathematics*, Springer, New York.

Bishop, E. (1967), *Foundations of Constructive Analysis*, Springer, New York.

Bishop, E. and D. Bridges (1985), *Constructive Analysis*, Springer, New York.

Bishop, E. and H. Cheng (1972), *Constructive Measure Theory,* Memoirs of the American Mathematical Society 116, AMS, Providence R. I.

Blum, L. (2004), Computability over the reals: Where Turing meets Newton, *Notices Amer. Math Soc.* 51, 1024-1034.

Blum, L., M. Shub and S. Smale (1989), On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines, *Bull. Amer. Math. Soc.* 21, 1-46.

Blum, L., F. Cucker, M. Shub and S. Smale (1997), *Complexity and Real Computation*, Springer, New York.

Braverman, M. and S. Cook (2006), Computing over the reals: Foundations for scientific computing, *Notices Amer. Math. Soc.* 51, 318-329.

Bridges, D. S. (1979), *Constructive Functional Analysis*, Pitman, London.

Bridges, D. S. and L. S. Vita (2006), *Techniques of Constructive Analysis*, Universitext, Springer-Verlag, Heidelberg.

Caviness B. F. and J. R. Johnson, eds. (1998), *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Springer, New York.

Chan, Y-K. (1974), Notes on constructive probability theory, *Annals of Probability* 2, 51-75.

Cutland, N. J. (1980), *Computability: An Introduction to Recursive Function Theory*, Cambridge Univ. Press, Cambridge.

Feferman, S. (1975), A language and axioms for explicit mathematics, in *Algebra and Logic* (J. N. Crossley, ed.), 87-139.

_____ (1979), Constructive theories of functions and classes, in *Logic Colloquium '78* (M. Boffa, et al., eds.), North-Holland, Amsterdam, 159-224.

_____ (1984), Between constructive and classical mathematics, in *Computation and Proof Theory* (M. M. Richter, et al., eds.), Lecture Notes in Computer Science 1104, 143-162.

_____ (1992a), A new approach to abstract data types, I: Informal development, *Mathematical Structures in Computer Science* 2, 193-229.

_____ (1992b), A new approach to abstract data types, II: Computability on ADTs as ordinary computation, in *Computer Science Logic* (E. Börger, et al., eds.), Lecutre Notes in Computer Science 626, 79-95.

_____ (1996), Computation on abstract data types: The extensional approach, with an application to streams, *Annals of Pure and Applied Logic* 81, 75-113.

Ferreira, F. (1994), A feasible theory for analysis, *J. Symbolic Logic* 59, 1001-1011.

Fischer, M. and M. Rabin (1974), Super-exponential complexity of Presburger arithmetic, in *Complexity of Computation*, AMS-SIAM Proceedings 7, 27-41; reprinted in Caviness and Johnson (1998), 122-135.

Friedman, H. (1971), Algorithmic procedures, generalized Turing algorithms, and elementary recursion theory, in *Logic Colloquium '69* (R. O. Gandy and C. M. E. Yates, eds.), North-Holland, Amsterdam, 361-389.

Friedman, H. and R. Mansfield (1992), Algorithmic procedures, *Transactions of the American Mathematical Society* 332, 297-312.

Grzegorczyk, A. (1955), Computable functionals, *Fundamenta Mathematicae* 42, 168-202.

Herman, G. T. and S. D. Isard (1970), Computability over arbitrary fields, *J. London Math. Soc.* 2, 73-79.

Kleene, S. C. (1952), *Introduction to Metamathematics*, North-Holland, Amsterdam.

Ko, K. (1991), *Complexity Theory of Real Functions*, Birkhäuser, Boston.

Ko, K. (1998), Polynomial-time computability in analysis, in *Handbook of Recursive Mathematics, Vol. 2, Recursive Algebra, Analysis and Combinatorics* (Yu. L. Ershov, et al., eds.), Elsevier, Amsterdam, 1271-1317.

Ko, K. and H. Friedman (1982), Computational complexity of real functions, *Theoretical Computer Science* 20, 323-352.

Lacombe, D. (1955), Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles, I, II, III, *Comptes Rendus de l'Académie des Science Paris*, 240: 2470-2480, 241: 13-14, 241: 151-155.

Mazur, S. (1963), Computable analysis, *Rozprawy Matematyczne* 33.

Mines, R., F. Richman and W. Ruitenberg (1988), *A Course in Constructive Algebra,* Springer, New York

Moldestad, J., V. Stoltenberg-Hansen and J. V. Tucker (1980a), Finite algorithmic procedures and inductive definability, *Mathematica Scandinavica* 46, 62-76.

_____ (1980b), Finite algorithmic procedures and inductive definability, *Mathematica Scandinavica* 46, 77-94.

Moschovakis, Y. N. (1984), Abstract recursion as a foundation for the theory of recursive algorithms, in *Computation and Proof Theory* (M. M. Richter, et al., eds.), Lecture Notes in Computer Science 1104, 289-364.

_____ (1989), The formal language of recursion, *J. Symbolic Logic* 54, 1216-1252.

Myhill, J. and J. C. Shepherdson (1955), Effective operations on partial recursive functions, *Zeitschr. f. Mathematische Logik u. Grundlagen der Mathematik* 1, 310-317.

Platek, R. A. (1966), *Foundations of Recursion Theory*, PhD Dissertation, Stanford University.

Pour-El, M. B. (1974), Abstract computability and its relation to the general purpose analog computer, *Trans. Amer. Math. Soc.* 199, 1-28.

Pour-El, M. B. and J. C. Caldwell (1975), On a simple definition of computable function of a real variable with applications to functions of a complex variable, *Zeitschr. f. Mathematische Logik u. Grundlagen der Mathematik* 21, 1-19.

Shepherdson, J. C. (1976), On the definition of computable function of a real variable, *Zeitschr. f. Mathematische Logik u. Grundlagen der Mathematik* 22, 391-402.

Shepherdson, J. C. and H. E. Sturgis (1963), Computability of recursive functions, *J. Assoc. Computing Machinery* 10, 217-255.

Spitters, B. (2003), *Constructive and Intuitionistic Integration Theory and Functional Analysis,* PhD Dissertation, University of Nijmegen.

Stoltenberg-Hansen, V. and J. V. Tucker (1999), Concrete models of computation for topological spaces, *Theoretical Computer Science* 219, 347-378.

Tarski, A. (1951), *A Decision Method for Elementary Algebra and Geometry*, Univ. of California Press, Berkeley; reprinted in Caviness and Johnson (1998), 24-84.

Tucker, J. V. and J. I. Zucker (1988), *Program Correctness over Abstract Data Types, with Error-State Semantics*, CWI Monographs 6, North-Holland, Amsterdam.

_____ (2000), Computable functions and semicomputable sets on many-sorted algebras, in *Handbook of Logic in Computer Science* Vol. 5 (S. Abramsky, et al., eds.), Oxford University Press, Oxford, 317-523.

_____ (2004), Abstract versus concrete computation on metric partial algebras, *ACM Transactions on Computational Logic* 5, 611-668.

_____ (2005), Computable total functions, algebraic specifications and dynamical systems, *J.l of Logical and Algebraic Programming* 62, 71-108.

Weirauch, K. (2000), *Computable Analysis*, Springer, Berlin.

Xu, J. and J. Zucker (2005), First and second order recursion on abstract data types, *Fundamenta Informaticae* 67, 377-419.

Ye, F. (2000), Toward a constructive theory of unbounded linear operators, *J. Symbolic Logic* 65, 357-370.