Theses for Computation and Recursion on Concrete and Abstract Structures

Solomon Feferman

Abstract: The main aim of this article is to examine proposed theses for computation and recursion on concrete and abstract structures. What is generally referred to as Church's Thesis or the Church-Turing Thesis (abbreviated **CT** here) must be restricted to concrete structures whose objects are finite symbolic configurations of one sort or another. Informal and principled arguments for **CT** on concrete structures are reviewed. Next, it is argued that proposed generalizations of notions of computation to abstract structures must be considered instead under the general notion of algorithm. However, there is no clear general thesis in sight for that comparable to **CT**, though there are certain wide classes of algorithms for which plausible theses can be stated. The article concludes with a proposed thesis **RT** for recursion on abstract structures.

1. Introduction. The concepts of recursion and computation were closely intertwined from the beginning of the efforts early in the 1930s to obtain a conceptual analysis of the informal notion of *effective calculability*. I provide a review of those efforts in sec. 2 as background to the remainder of this article, but I have nothing new to add here to the extensive historical and analytical literature.[1] It is generally agreed that the conceptual analysis of effective calculability was first provided most convincingly by Turing (1936-7). Not long before that Church (1936) had proposed identifying effective calculability with the Herbrand-Gödel notion of general recursiveness, soon enough proved equivalent to Turing computability among other suggested explications. Curiously, the subject of effective computation came mostly to be called *Recursion Theory*, even though that is only one of the possible forms of its development.[2]

In his influential book, *Introduction to Metamathematics*, Kleene (1952) baptized the statement that every effectively calculable function is general recursive as *Church's*

[1] Gandy (1988) is an excellent introductory source to those developments; cf. also Soare (1999).

[2] Soare in his articles (1996, 1999) has justifiably made considerable efforts to reconfigure the terminology of the subject so as to emphasize its roots in the notion of computation rather than recursion, for example to write 'c.e.' for 'computably enumerable' in place of 'r.e.' for 'recursively enumerable', but they do not seem to have overcome the weight of tradition.

Thesis. He went on to baptize as *Turing's Thesis* the statement that "every function which would naturally be regarded as computable is computable …by one of his machines"[3] and to say that this is equivalent to Church's Thesis, since the general recursive functions are exactly the same as the (total) functions computable by Turing machines. This led Kleene to speak further on in his book in ambiguous terms of the *Church-Turing Thesis*. Among workers in recursion theory it is common to take Church's Thesis and Turing's Thesis to be equivalent for the same reason as given by Kleene, and to follow him in referring to them without distinction as Church's Thesis or as the Church-Turing Thesis; I shall also use '**CT**' ambiguously as an abbreviation for either of these.[4]

My main concern in this article is to examine proposed theses for computation and recursion on both concrete and abstract structures. By *concrete structures* I mean those given by sets of finite symbolic configurations (e.g., finite strings, trees, graphs, hereditarily finite sets, etc.) together with the appropriate tests and operations for their transformation. In sec. 3 I review some efforts to "prove" **CT** for computation on concrete structures that began with Gandy (1980) and were later pursued by Sieg and by Dershowitz and Gurevich among others (references below). The approaches in question proceed by isolating basic properties of the informal notion of effective calculability or computation in axiomatic form and proving that any function computed according to those axioms is Turing computable. It is not my aim here to argue for one or another of these approaches but rather to emphasize what they hold in common (and what is usually taken for granted), namely that *in whatever way one understands **CT**, there is no calculation without representation*, i.e. it is a necessary ingredient of whatever constitutes effective calculability that one operates only on finite symbolic configurations by means that directly transform such configurations into new ones using appropriate tests along the way.

---

[3] Kleene's wording derives from Turing (1936-7), p. 249: "No attempt has yet been made to show that the 'computable' numbers include all numbers *which would naturally be regarded as computable*." [Italics mine]

[4] Cf. Copeland (2002) for an introductory article on the Church-Turing thesis.

Beginning in the late 1950s a number of generalizations of the notions of computation and recursion were made to a great variety of *abstract structures*, including general first-order (or algebraic) structures, finite-type structures over the natural numbers, and structures of sets. These developments witnessed impressive success in obtaining various analogues of leading results from classical recursion theory. Nevertheless, the point of my emphasis on concrete structures as a *sine qua non* for **CT** is to raise questions about proposed generalizations of it to abstract structures that have been suggested. In particular, I shall concentrate in sec. 4 on general theories of "computation" on first-order structures that descend from Friedman (1971) via his adaptation of the Shepherdson-Sturgis register machine approach (equivalently, the Turing machine approach) on the one hand, and that of Tucker and Zucker (1988, 2000) via "While" schemata on the other. I shall argue (as Friedman already did) that the proposed generalizations are more properly examined under the concept of *algorithmic procedures*, which are meaningful for abstract structures (or "data types") since algorithms are independent of the means by which individual data items and the operations on them may be represented. As will be explained at the end of sec. 4, substantial efforts have been made to answer the question, "What is an algorithm?," leading to quite different conclusions, among them by Moschovakis (1984, 2001) and Gurevich (2012). In view of the controversy, it is perhaps premature to propose a general associated version of **CT** for algorithms, though wide classes of algorithms may be candidates for such. In particular, Tucker and Zucker (op. cit.) have formulated a thesis for algebraic algorithmic procedures on abstract structures that deserves special attention under that heading.

Finally, by comparison with these, sec. 5 reviews a general notion of recursive definition applied to suitable second-order structures, and a general thesis **RT** related to such is proposed for consideration.

Before turning to this material, one may well ask: What purpose is served by the questions of formulation and justification of these theses? We should not expect a single answer in each case, let alone a single overall answer; here are a few. In the case of **CT** itself, we had the historical pressure on the negative side to demonstrate the effective

unsolvability of the *Entscheidungsproblem* in logic and, subsequently, many more examples of that in mathematics. On the positive side, it was and continues to be used to provide a solid foundation to informal proofs of computability, i.e. to justify "proofs by Church's Thesis." (And for Gödel, it was used to bolster his conviction that mind is not mechanical, via the incompleteness theorems and the identification of formal systems in their most general form with Turing machines.) Moving on to the proposed extension of notions of computability to abstract structures, a prime structure of interest is that of the real numbers (and relatedly the complex numbers), the arena of numerical analysis (aka scientific computation). Among the concerns here are to understand its limits and to provide a unified foundation. Moreover, as we shall see, a crucial issue is to separate conceptually algebraic methods from analytical ones. In addition, the latter are relevant to questions as to whether and in what sense the laws of physics are mechanical.

More generally, it is important to separate and refine concepts that are often confused, namely those of *computation procedure*, *algorithmic procedure* and *recursive definition* that are the primary concern of this article. Those who are philosophically inclined should find interest in these case studies in conceptual analysis (aka explication of informal concepts) both settled and unsettled. Finally, it may be hoped that satisfactory progress on these questions would help lead to the same on concepts of *feasibility* in these various areas, concepts that are wholly untouched here.

2. Recursion and computation on the natural numbers, and the Church-Turing Thesis. The question as to which forms of definitions of functions on the natural numbers are finitistically acceptable was an important issue in the development of Hilbert's program in the 1920s. In the views of the Hilbert school this certainly included the primitive recursive functions (going back to Dedekind), but was not limited to such once Ackermann produced his example of a non-primitive recursive function. Formally, that was given by a nested double recursion, but it could also be analyzed as a recursion on the ordinals up to ω^2. Hilbert's unsupported claim (1926) to have demonstrated the Continuum Hypothesis by means of his proof theory involved the use of functions given by higher transfinite recursions; it would thus seem that Hilbert counted them among the

finitistically acceptable functions, though no clear conditions were given as to what would make them so.

A more specific analysis of which functions of the natural numbers are definable by recursive means came about as a result of a single exchange of correspondence between Herbrand and Gödel in 1931. Herbrand wrote first after receiving a copy of Gödel's paper on the incompleteness of arithmetic that happened to make essential technical use of the schemata for primitive recursive functions. Concerning the contents of that letter, in 1963 Jean van Heijenoort asked Gödel about the exchange in connection with his (1934) formulation of the notion of general recursive function, the idea for which he had credited in part to Herbrand. In his response to van Heijenoort, Gödel said that he could not locate the correspondence in his papers but that the formulation in the 1934 notes was exactly as had been proposed to him three years prior to that by Herbrand. The two letters in the exchange continued to be missing in the following years until they were found finally by John W. Dawson, Jr. in 1986 in the course of his cataloguing of the Gödel *Nachlass* at the Institute for Advanced Study in Princeton. As explained in detail in Dawson (1993) and Sieg (2003, 2005) it turned out that Gödel misremembered some essential points.[5]

Herbrand's letter to Gödel informally proposed a characterization of the extent of the finitistically acceptable functions in terms of recursive definitions via a single formal system of arithmetic with quantifier-free induction and axioms for a sequence of functions f_n satisfying three conditions on successive quantifier free axioms for the f_n, of which the main one is:

> We must be able to show, by means of intuitionistic [finitary] proofs, that with these axioms it is possible to compute the value of the functions univocally for each specified system of values of their arguments. (cf. Sieg 2003, p. 6)

Among examples, Herbrand lists the recursion equations for addition and multiplication, Gödel's schemata for primitive recursion, Ackermann's non-primitive recursive function, and functions obtained by diagonalization. Gödel's reply, though friendly, was critical

[5] For convenience I shall only refer to Sieg (2003) in the following, though there is considerable overlap with Dawson (1993).

on a number of points, among which Herbrand's proposal that finitism could be encompassed in a single formal system.[6]

In the notes for his 1934 lectures on the incompleteness theorems at the IAS, Gödel returned to Herbrand's proposal in the last section under the heading "general recursive functions." He there recast Herbrand's suggestion to define a new function φ in terms of "known" functions $\psi_1,\ldots,\psi_k$ and possibly φ itself. The main requirement is now taken to be that for each set of natural numbers $k_1,\ldots,k_l$ there is one and only one m such that $\varphi(k_1,\ldots,k_l) = m$ is a derived equation, where the rules of inference for the notion of derivation involved are simply taken to be those of the equation calculus, i.e. substitution of numerals for variables and substitution of equals for equals (cf. Gödel 1934, p. 26).

Before turning to this bridge to the conceptual analysis of effective calculability, note that there are two features of Gödel's definition of general recursive function that make it ineffective in itself, namely (given "known" effectively calculable functions $\psi_1,\ldots,\psi_k$) one can't decide whether or not a given system of equations E has (i) the uniqueness property, namely that for each sequence of arguments $k_1,\ldots,k_l$ there is at most one m such that $\varphi(k_1,\ldots,k_l) = m$ is derivable from E, nor that E has (ii) the existence property, namely that there is at least one such m for each such sequence of arguments. In Kleene's treatments of general recursion elaborating the Herbrand-Gödel idea, beginning in 1936 and ending in Kleene (1952) Ch. XI, one may use *any* system of equations E to determine a partial recursive function, simply by taking $\varphi(k_1,\ldots,k_l)$—when defined—to be the value m given by the least derivation (in a suitable primitive recursive coding) ending in some value for the given arguments. It is of course still undecidable whether the resulting function is total. It finally remained for Kleene to give the most satisfactory general formulation of effective recursion on the natural numbers via his Recursion Theorem (op. cit., p. 348). But this requires the notion of partial recursive functional to which we shall return in sec. 5 below.

[6] Gödel was not long after to change his mind about that; cf. Gödel (1933).

Now, not only did Gödel in 1934 misremember the details of Herbrand's 1931 formulation, he made a crucial conceptual shift there from the question of characterizing the totality of finitistically acceptable functions to that of characterizing the totality of functions given by a "finite computation" procedure, despite his clear reservations both about the possibility of such and of general recursion in particular as a prime candidate. Church was bolder:

> We now define [sic!] the notion … of an effectively calculable function of positive integers by identifying it with the notion of recursive function of positive integers (or of a λ-definable function of positive integers). (Church 1936, p. 356)

Church had previously proposed to identify the effectively calculable functions with the λ-definable ones, but by 1936 he could depend on their co-extensiveness with the general recursive functions established in Kleene (1936, 1936a). It was Kleene (1943) who baptized Church's "definition" of effectively calculable function as 'Thesis I' and then as Church's Thesis in Kleene (1952), p. 300. See Gandy (1988) pp. 76 ff for Church's full statement of the Thesis (*qua* definition) and the initial arguments that he made in its favor, among which the equivalence of "two such widely different and … equally natural definitions of effective calculability"; Gandy calls that *the argument by confluence of ideas.*[7]

This argument was soon to be extended by the most significant step in the analysis of the concept of effectively calculable function on the natural numbers, namely that made by Turing (1936-37). To be noted is that Turing conceives of the calculations as being carried out by an (abstract) human being following a fixed finite set of instructions or routine using a finite set of symbols specified in advance, via entering or deleting the contents of the workspace given by a potentially infinite set of symbol locations or cells. The confluence argument was bolstered by Turing's (1937) proof of the equivalence of his notion of computability with that of λ-definability. Church quickly accepted Turing's analysis of effectively calculable as the preferred notion of the three then available. As he wrote in his review of Turing's paper:

[7] Church (1936) pp. 100-102 had another "step-by-step" argument for the Thesis, but there is a semi-circularity involved; cf. Shagrir (2002) p. 224.

> [Turing's notion] has the advantage of making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately—i.e., without the necessity of proving preliminary theorems. (Church 1937)

Gödel, too, accepted Turing's explication of effective calculability, but by when is not clear. In the unpublished lecture designated **193?* in Gödel (1995)—probably prepared in 1938—he describes the notion of general recursive function, and then writes, "[t]hat this really is the correct definition of mechanical computability was established beyond any doubt by Turing" (Gödel 1995, p. 168); however, he does not refer to proofs of their equivalence as justification for this. The first evidence of his view in print appears briefly at the beginning of his remarks before the 1946 Princeton Bicentennial, where he speaks of "the great importance of the concept of general recursiveness (or Turing's computability)" because of the independence of the concept of calculability in a formal system from the formalism chosen (cf. Gödel 1990, p.150). But again he does not take Turing's explanation of effective computability to be the primary one. Only later, in his June 1964 Postscript to Gödel (1934), does he address the concept on its own terms as follows:

> Turing's work gives an analysis of the concept of "mechanical procedure" (alias "algorithm" or "computation procedure" or "finite combinatorial procedure"). This concept is shown to be equivalent with that of a "Turing machine". (cf. Gödel 1986, p. 370)

As described in sec. 1 above, it was Kleene (1952) p. 300 *et seq*, who led one to talk of Church's Thesis, Turing's Thesis, and then, ambiguously, of the Church-Turing Thesis for the characterization through these equivalences of the effectively calculable functions. In another influential text, Rogers (1967) pp. 18ff, took the argument by confluence as one of the basic pins for **CT** and used that to justify informal proofs "by Church's Thesis".

3. Computation on concrete structures and "proofs" of CT. Beginning with work of Kolmogorov in 1953 (cf. Kolmogorov and Uspensky 1953), efforts were made to move beyond the argument by confluence, among others, to more principled arguments for **CT**. The first significant step in that direction was made by Gandy (1980) which, together

with its successors to be described below, may be construed as following a more axiomatic approach, in the sense that one (cl)aims to isolate basic properties of the informal notion of effective calculability or computation and proves that any function computed according to those conditions is computable by a Turing machine. As one sees by closer inspection, all the axioms used by the work in question take the notion of finiteness for granted, hence may be construed formally as carried out within weak second-order logic, but otherwise there is considerable difference as to how they are formulated.

To begin with, Gandy asserts (as he did again in Gandy (1988)) that Turing outlined a proof of the following in his famous paper (1936-7):

Thesis T. What can be calculated by an abstract human being working in a routine way is computable [by a Turing machine].

Actually, such a statement, being non-mathematical, can't be proved, nor did Turing claim to have done so. Rather, what he *did* do in sec. 9 of the paper was to present three informal arguments as to why his analysis catches everything that "would naturally be regarded as computable;" it is the first of these arguments that leads most directly to the concept of a Turing machine. The argument in question sets out five informal restrictive conditions on the idealized work space and possible actions within it of a human computer. As recast by Sieg (2002, 2002a), in order to proceed to a theorem there are two steps involved: first, Turing's conditions are reformulated more generally in terms of boundedness, locality and determinacy conditions (still at the informal level), and, secondly, those conditions are given a precise mathematical expression for which it can be shown that any function satisfying them is computable by a Turing machine. (Sieg calls the second part a representation theorem.)

By contrast to Thesis T, the aim of Gandy's 1980 article was to argue for the following:

Thesis M. What can be calculated by a machine is computable [by a Turing machine].

Again, there is no proof of Thesis M, but rather a two part procedure that eventuates in a definite theorem. The first part consists of Gandy's informal restrictions on what constitute "mechanical devices" (1980, pp. 125-126), namely, that he excludes from consideration devices that are "*essentially* analogue machines" and that "[t]he only physical presuppositions made about mechanical devices … are that there is a lower bound on the linear dimensions of every atomic part of the device and that there is an upper bound (the velocity of light) on the speed of propagation of changes." Furthermore, Gandy assumes that the calculations by a mechanical device are describable in discrete terms and that the behavior of the device is deterministic, though calculations may be carried out in parallel. These restrictions then lead to the formulation of four mathematically precise principles I-IV that express the informal conditions on what constitutes a machine in his sense. The main result of Gandy (1980) is a theorem to the effect that any function calculated by a mechanical device satisfying principles I-IV is computable on a Turing machine.

By the way, in the informal part of his argument for **Thesis** M, Gandy enlarges on the discreteness aspect in a way that is particularly useful for our purposes below.

> Our use of the term "discrete" presupposes that each state of the machine can be adequately described in finite terms. … [*W*]*e want this description to reflect the actual, concrete, structure of the device in a given state.* On the other hand, we want the form of the description to be sufficiently abstract to apply uniformly to mechanical, electrical or merely notional devices. We have chosen to use hereditarily finite sets; other forms of description might be equally acceptable. We suppose that the labels are chosen for the various parts of the machine—e.g., for the teeth of cog wheels, for a transistor and its electrodes, for the beads and wires of an abacus. Labels may also be used for positions in space (e.g., for squares of the tape of a Turing machine) and for physical attributes (e.g., the color of a bead, the state of a transistor, the symbol on a square). (Gandy 1980 p. 127, italics mine)

In other words, just as with **Thesis T**, one is working throughout with finite symbolic configurations.

Gandy's case for **Thesis M** was substantially recast by Sieg (2002, 2002a) much as he had done for **Thesis T**. The first part of that work is again that in carrying out effective calculations, the machine is limited by general boundedness, locality and

determinacy conditions, but those are now widened to allow acting on given finite configurations in parallel and then reassembling the results into the next configuration. That led Sieg to a statement of new simpler precise principles on mechanisms as certain kinds of discrete dynamical systems for which a representation theorem is proved, i.e. for which it is shown that whatever satisfies those principles "computes" only Turing computable functions.

Yet another approach toward establishing a version of **CT** for certain kinds of mechanisms is that due to Dershowitz and Gurevich (2008), entititled "A natural axiomatization of computability and proof of Church's Thesis."[8] The mechanisms in question are called Abstract State Machines (ASMs); fundamental to this work is that of Gurevich (2000) in which it is argued that sequential algorithms are captured by Sequential ASMs. Regrettably, the statement by Dershowitz and Gurevich of Church's Thesis is not the one usually understood but rather their Theorem 4.8, according to which "[e]very numeric partial function computed by an arithmetical algorithm is (partial) recursive." And for this, arithmetical algorithms are defined as state transition systems satisfying certain Sequential and Arithmetical Postulates. Unfortunately, the postulates are not fully precise as presented, and so it may be questioned whether one even has a proof there of a definite mathematical theorem.[9] One point to be noted for the following is that the ASM approach takes states to be (abstract) structures having a common fixed finite vocabulary, and a crucial assumption thereto is Postulate III (p. 319) according to which state transitions are "determined by a fixed finite 'glossary' of 'critical terms'." Thus, though the terminology makes it seem otherwise, Abstract State Machines work concretely with suitable finite symbolic configurations in their computational processes; cf. also op. cit., Definition 3.1, p. 321.

The work by Gandy, Sieg, Dershowitz and Gurevich described in the preceding must be valued for taking seriously the task of providing a two part argument for **CT** mediated by axioms of one form or another. However, it may be questioned whether the

[8] Dershowitz and Gurevich (2008) p. 305 state that the aim of their work is "to provide a small number of convincing postulates in favor of Church's Thesis"; in that same article, pp. 339-342, they provide a comprehensive survey of the literature sharing that aim, going back to work of Kolmogorov in 1953.

[9] Cf. the Postscriptum to Sieg (2013) for a detailed critique of this work.

axioms provided for any of these yet reaches the desired degree of evidence, in other words of being close to compelling on inspection. But what is common to these axiomatic approaches and cannot be denied is that the *sine qua non* of **CT** is that *there is no calculation without representation.* That is, the data with which one works consists of finite symbolic configurations where the symbols (or labels) are drawn from some finite set S given in advance. These represent finite concrete configurations such as finite linear inscriptions by human beings, or mathematical configurations such as finite trees or graphs, or states of various kinds of mechanisms such as described in the quote above from Gandy (1980) p. 127. More abstractly, Gandy considered finite symbolic configurations to be themselves represented in the hereditarily finite non-empty sets over the basic set S of symbols, though I think representation in the hereditarily finite non-empty sequences over S would be more appropriate since that gives an order in which things must be read; of course, each can be coded in the other. The operations on finite symbolic configurations must be limited to purely formal transformations following inspections and appropriate tests. Thus the claim here is that a general discussion of **CT** as it applies to computation over arbitrary structures *only* makes sense when applied to computation over concrete structures whose elements are finite symbolic configurations of one sort or another and that posit appropriate tests and operations on such.[10]

4. Proposed generalizations of theories of computation and CT to abstract structures; theses for algorithms. Beginning in the late 1950s, various generalizations were made of theories of computation and recursion theory to abstract structures in general and of certain specific kinds of structures.[11] My concern here is entirely with the foundations of such approaches, not with the results obtained on their basis. The article Feferman (2013) contains a more extensive exposition of these matters; the reader is referred to that for more details. What is discussed here—but not there—are proposed generalizations of **CT** to structures that need not be concrete. I shall consider two such

[10] For a systematic treatment of computability on concrete structures see Tucker and Zucker (2006).

[11] The literature on generalized recursion theory is very extensive and could use an up-to-date survey. Lacking that, some initial sources can be found in the bibliographies in the works of Barwise (1975), Fenstad (1980), Sacks (1990), and Tucker and Zucker (2000).

below, one (implicitly) due to Blum, Cucker, Shub and Smale (1997) and the other (explicitly) due to Tucker and Zucker (1988, 2000). I shall argue that these are more appropriately to be viewed as theses for algorithms.

An important starting point is the notion of computability on an arbitrary algebraic structure made by Friedman (1971) via a generalization of the Shepherdson and Sturgis (1963) register machine approach to ordinary recursion theory. By a (first-order) structure or algebra $\mathfrak{A}$ is meant one of the form $\mathfrak{A} = (A, c_1,\ldots,c_j, f_1,\ldots,f_k, R_1,\ldots,R_m)$, 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 (possibly partial) relation of one or more arguments in A. For non-triviality, both k and m are not zero. 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 among which one is designated as initial and one as terminal. The "machine" has registers r_0, r_1, r_2, …, though only a finite number of these 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 n. Given an input $(x_1, \ldots, x_n)$, one enters x_i into register r_i, and proceeds to the initial instruction. The active instructions are: (i) replace the content of one register by that of another; (ii) enter one of the c_i in a specified register; (iii) enter a value of one of the f_i applied to the contents of specified registers into another such; and, finally, (iv) test one of the R_i on specified registers and go to designated other instructions depending on the value of the test ("if …then… else"). The computation terminates only if the instructions of the form (iii) and (iv) are defined at each stage where they are called and one eventually lands in the terminal instruction. In that case the content 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**($\mathfrak{A}$). Friedman (1971) also gives an extensionally equivalent formulation of computability on $\mathfrak{A}$ in terms of generalized Turing machines, as well as one in terms of

what he calls effective definitional schemata given by an effective infinite enumeration of definition by cases.

For the structure $\mathfrak{N}$ = (**ℕ**, 0, Sc, Pd, =), where 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 determined in this way.

The notion of finite algorithmic procedure is directly generalized to many-sorted structures $\mathfrak{A} = (A_1,\ldots,A_n, c_1,\ldots,c_j, f_1,\ldots,f_k, R_1,\ldots,R_m)$; 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, Stolenberg-Hansen and Tucker (1980, 1980a), using *stack registers* that 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 to remove the top element of a stack (*pop*) and to add to the contents of one of the registers of type A_i (*push*). This leads to the notion of what is computable by a *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 jth 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

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

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

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

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

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

An alternative approach to computability over arbitrary algebraic structures is provided in a usefully detailed expository piece, Tucker and Zucker (2000) that goes back to their joint work (1988); this uses definition by schemata rather than (so-called) machines. 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. The Tucker-Zucker notion of computability for standard algebras is given by procedure statements S: these include explicit definition, and are closed under composition, and under statements of the form, *if* b *then* S_1 *else* S_2, and *while* b *do* S, where 'b' is a Boolean term. The set of partial functions computable on $\mathfrak{A}$ by means of these schemata is denoted by $\mathbf{While}(\mathfrak{A})$. 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 suitable $\mathfrak{A}^*$. The notions of computability $\mathbf{While}^{\mathfrak{N}}(\mathfrak{A})$ and $\mathbf{While}^*(\mathfrak{A})$ over $\mathfrak{A}$ are given simply by $\mathbf{While}(\mathfrak{A}, \mathfrak{N})$ and $\mathbf{While}(\mathfrak{A}^*)$, respectively. The following result is stated in Tucker and Zucker (2000) p. 487 for any standard algebra $\mathfrak{A}$:

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

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

Thus we have a certain robustness (confluence of ideas) for notions of computation on abstract algebraic structures, depending on the choice as to whether or not to include the natural numbers or finite sequences.

The first interesting special non-concrete case to which these notions may be applied is the structure of real numbers; this is of particular significance because it is the principal domain for numerical analysis (aka scientific computation). One approach to the foundations of that subject is given by a model of computation over the reals due to Blum,

Shub and Smale (1989)—the BSS model—subsequently worked out at length in the book, Blum et al. (1997). Actually, the model is divided into two cases, the finite dimensional one and the infinite dimensional one. In the first of these, the reals are treated as a purely algebraic structure, namely the ordered field $\mathfrak{R} = (\mathbb{R}, 0, 1, +, -, \times, ^{-1}, <)$, while in the second case, one also computes with arbitrary finite sequences of reals. According to Friedman and Mansfield (1992) p. 298, in the finite dimensional case the BSS computable functions are exactly the same as the **FAP**($\mathfrak{R}$) functions, and in the infinite dimensional case the BSS computable functions are exactly the same as the **FAPS**($\mathfrak{R}$) functions. Moreover, one also has **FAPS**($\mathfrak{R}$) = **FAPCS**($\mathfrak{R}$), because $\mathfrak{N}$ can be embedded in $\mathfrak{R}$ (op. cit., p.300). Note that the relations of equality and order on the reals are an essential part of the BSS model, as is equality in general for all algebraic structures in that model.

The case made in Blum et al. (1997) for the extension of the "classical" notion of computation to computation on the reals via the BSS model is not one argued on its own merits but rather mainly by its claimed requisite applicability. This follows a brief review of theories of computation for concrete structures (the subject of "computer science") as well as Church's Thesis, whose support is bolstered by the confluence of notions. Then the authors say that "[c]ompelling motivation clearly would be required to justify yet a new model of computation" (op. cit., p. 22). And that is claimed to come from a need to give foundations to the subject of numerical analysis:

> A major obstacle to reconciling scientific computation and computer science is the present view of the machine, that is, the digital computer. As long as the computer is seen as a finite or discrete object, it will be difficult to systematize numerical analysis. We believe that the Turing machine as a foundation for real number algorithms can only obscure concepts. Toward resolving the problem we have posed, we are led to expanding the theoretical model of the machine to allow real numbers as inputs. (Op. cit., p. 23)

An analogy (pp. 23-24) is made with Newton's problem of reconciling the discrete corpuscular view of matter that he accepted with the mathematics of the calculus that he found necessary to describe bodies in *prima facie* continuous motion; the resolution came via idealized infinitesimal masses.

> Now our suggestion is that the modern digital computer could be idealized in the same way that Newton idealized his discrete universe. … Moreover, if one regards computer-graphical output such as our picture of the Mandelbrot or Julia sets with their apparently fractal boundaries and asks to describe the machine that made these pictures, one is driven to the idealization of machines that work on real or complex numbers in order to give a coherent explanation of these pictures. For a wide variety of scientific computations the continuous mathematics that the machine is simulating is the correct vehicle for analyzing the operation of the machine itself.
>
> These reasonings give some justification for taking as a model for scientific computation a machine model that accepts real numbers as inputs. (Op. cit., p. 24)

What is puzzling in this analogy is that on the BSS model of computation, the relation of order and hence that of equality between real numbers is taken as total and decidable by the idealized machine, and so one is immediately led to discontinuous functions, such as point and step functions. Moreover, the BSS model makes use only of the algebraic structure of the real numbers and nothing that directly reflects its analytic/topological character. So it fails to provide a genuine notion of computation on the real numbers *as such* for which a version of **CT** would be claimed to hold. Nevertheless, as illustrated by a number of leading examples, Blum et al. (1997) makes a substantial case that the BSS model provides a proper foundation for the subject of numerical analysis where the basic data is taken to be given by real (or complex) numbers. Why this turns out to be so is a matter to which I shall return at the beginning of the next section.

The question whether there is a sensible generalization of the Church-Turing Thesis to abstract structures is addressed directly by Tucker and Zucker (1988), pp. 196ff and again in Tucker and Zucker (2000), pp. 493ff. In the latter it is said that the answer to the question is difficult to explain fully and briefly so that only a sketch is given, and the reader is referred back to the former for more details. Though the later publication is a bit more succinct than the earlier one on this issue, I didn't find that it leaves out any essential points, so I shall use that as the reference in the following.[12] The authors begin with the statement of a "naïve" generalized **CT** for abstract algebras, namely that "[t]he

[12] In addition, the reference Tucker and Zucker (1988) is not as widely available as their year 2000 survey.

functions that are 'effectively computable' on a many-sorted algebra $\mathfrak{A}$ are precisely the functions that are **While*** computable on $\mathfrak{A}$." This is immediately qualified as follows:

> [T]he idea of effective calculability is complicated, as it is made up from many philosophical and mathematical ideas about the nature of finite computation with finite or concrete elements. For example, its analysis raises questions about the mechanical representation and manipulation of finite symbols; about the equivalence of data representations; and about the formalization of *constituent concepts* such as *algorithm*; *deterministic procedure*; *mechanical procedure*; *computer program*; *programming language*; *formal system*; *machine*; and the functions definable by these entities. … However, only *some* of these constituent concepts can be reinterpreted or generalized to work in an abstract setting; and hence the general concept, and term, of 'effective computability' does not belong in a generalization of the Church-Turing thesis. In addition, since finite computation on finite data is truly a fundamental phenomenon, it is appropriate to preserve the term with its established special meaning. (Tucker and Zucker (2000), p. 494, italics in the original.)

In other words, these authors and I are in complete agreement with the view asserted at the end of the preceding section. Nevertheless, they go on to formulate three versions of a generalized **CT** *not* using the notion of effective calculability, corresponding to the three perspectives of algebra, programming languages, and specification on data types; only the first of these is relevant to the discussion here. Namely:

> **Tucker-Zucker thesis for algebraic computability.** The functions computable by finite deterministic algebraic algorithms on a many-sorted [first-order] algebra $\mathfrak{A}$ are precisely the functions While* computable on $\mathfrak{A}$. (op. cit., p. 495)

This goes back to the work of Tucker (1980) on computing in algebraic structures; cf. also Stoltenberg-Hansen and Tucker (1999). Hermann's algorithm for the ideal membership problem in $K[x_1,\ldots,x_n]$ for arbitrary fields K is given as a paradigmatic example, but there is no principled argument for this thesis analogous to the work of Gandy, Sieg, Dershowitz and Gurevich described in the preceding section. One may ask, for example, why the natural number structure and arrays are assumed in the Tucker-Zucker Thesis, and why these suffice beyond the structure $\mathfrak{A}$ itself. Moreover, nothing is said about assuming that the equality relation for $\mathfrak{A}$ is to be included in it, even though that is common in algebraic algorithms. Finally, one would like to see a justification of

this thesis or possible variants comparable to the ones described for classical **CT**, both informal and of a more formal axiomatic kind.

In any case, the Tucker-Zucker Thesis and supporting examples suggest that *all the notions of computability on abstract first order structures considered in this section should be regarded as falling under a general notion of algorithm.* What distinguishes algorithms from computations is that they are independent of the representation of the data to which they apply but only require how data is packaged structurally, i.e. they only need consider the data up to structural isomorphism. Friedman was already sensitive to this issue and that is the reason he gave for baptizing his notion using generalized register machines, *finite algorithmic procedures*:

> The difference between [symbolic] configuration computations and algorithmic procedures is twofold. Firstly, in configuration computations the objects are symbols, whereas in algorithmic procedures the objects operated on are unrestricted (or unspecified). Secondly, in configurational computations at each stage one has a finite configuration whose size is not restricted before computation. On the other hand in algorithmic procedures one fixes beforehand a finite number of registers to hold the objects. Thus for some n, at each stage one has at most n objects. (Friedman 1978, p. 362).

The general question, "What is an algorithm?" has been addressed by Moschovakis (2001) and Gurevich (2012) (both under that title), among others, but with very different conclusions.[13] In his sec. 6, Gurevich criticizes Moschovakis' answer on several grounds among which that distributed algorithms do not fall under the latter's central notion of *recursor*. Moreover, even those algorithms that fall under the notion of recursor may do so by losing certain essential aspects of the procedure in question. Whether or not one agrees with all of Gurevich's critiques of Moschovakis' analysis, in my view that is more appropriately to be considered under general theses for recursion that are taken up in the next section. In contrast to Moschovakis, Gurevich asks whether the notion of algorithm can be defined at all; his answer is "yes and no". On the negative side, he writes:

> In our opinion, the notion of algorithm cannot be rigorously defined in full generality, at least for the time being. The reason is that the notion is expanding.

[13] See also Blass and Gurevich (2003).

> Concerning the analogy of algorithms to real numbers, mentioned in sec.1, Andreas Blass suggested a better analogy: algorithms to numbers. Many kinds of numbers have been introduced throughout history: positive integers, natural numbers, rationals, reals, complex numbers, quaternions, infinite cardinals, infinite ordinals, etc. Similarly many kinds of algorithms have been introduced. In addition to classical sequential algorithms, in use from antiquity, we have now parallel, interactive, distributed, real-time, analog, hybrid, quantum, etc. algorithms. New kinds of numbers and algorithms may be introduced. The notions of numbers and algorithms have not crystallized (and maybe never will) to support rigorous definitions. (Gurevich 2012, sec. 2)

On the positive side he says that even though it is premature to try to propose a general answer to the question, "What is an algorithm?," convincing answers have been given to large classes of such, among which sequential algorithms, synchronous parallel algorithms and interactive sequential algorithms (cf. ibid for references). In particular, the Tucker-Zucker Thesis or something close to it is a plausible candidate for what one might call the *Algebraic Algorithmic Procedures Thesis*. And more generally, it may be possible to distinguish algorithms used in pure mathematics from those arising in applied mathematics and computer science, where such algorithms as "interactive, distributed, real-time, analog, hybrid, quantum, etc." would fall. If there is a sensible separation between the two, Moschovakis' explanation of what is an algorithm could be justified as being confined to those of pure mathematics. Moreover, his theory leads to the remarkable result that there is a decidable criterion for identity of algorithms in his sense (Moschovakis 1989).

Clearly, all this requires deeper consideration, and I must leave it at that.

5. Recursion on abstract structures. Let us return to the claim of Blum et al. (1997) that the BSS model of computation on the reals (and complex numbers) is requisite for the foundations of the subject of scientific computation. That was strongly disputed by Braverman and Cook (2006), where the authors argued that the requisite foundation is provided by a quite different "bit computation" model that is *prima facie* incompatible with the BSS model. It goes back to ideas due to Banach and Mazur in the latter part of the 1930s, but the first publication was not made until Mazur (1963). In the meantime, the bit computation model was refined and improved by Grzegorczyk (1955) and

independently by Daniel Lacombe (1955) in terms of a theory of recursively computable functionals. Terminologically, something like "effective approximation computability" is preferable to "bit computability" as a name for this approach in its applications to analysis.

This competing approach was explained in Feferman (2013) in rough terms as follows. To show that a real valued function f on a real interval into the reals is computable by effective approximation, given any x in the interval as argument to f, one works *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. The sequences in question are functions from $\mathbb{N}$ to $\mathbb{Q}$, and so what is required is that the passage from $\langle q_n \rangle_{n \in \mathbf{N}}$ to $\langle r_m \rangle_{m \in \mathbf{N}}$ is given by an effective type-2 functional on such functions. Write $\mathcal{T}$ for the class of all total functions from $\mathbb{N}$ to $\mathbb{N}$, and $\mathcal{P}$ for the class of all partial functions from $\mathbb{N}$ to $\mathbb{N}$. By the effective enumeration of the rational numbers, this reduces the notion of effective approximation computability of functions f on the reals to that of effective functionals F from $\mathcal{T}$ to $\mathcal{T}$, and those in turn are restrictions to $\mathcal{T}$ of the partial recursive functionals F' (from $\mathcal{P}$ to $\mathcal{P}$) whose values on total functions are always total.[14] It may be shown that by the continuity in the recursion theoretic sense of partial recursive functionals we may infer continuity in the topological sense of the functions f on the reals that are effective approximation computable. Thus step functions that are computable in the BSS model are not computable in this sense. On the other hand, the exponential function is an example of one that is computable in the effective approximation model that is not computable in the BSS model.[15]

[14] Note that a partial recursive functional F need not have total values when restricted to total arguments.

[15] There is a considerable literature on computation on the real numbers under various approaches related to the effective approximation one via Cauchy representations. A more comprehensive one is that given by Kreitz and Weihrauch (1984, 1985) and

The reader must be referred to Blum et al. (1997) and Braverman and Cook (2006) for arguments as to which, if either of these, is the appropriate foundation for scientific computation.[16] I take no position on that here, but simply point out that we have been led in a natural way from computation on the reals in the effective approximation sense back to the partial recursive functionals F on partial functions of natural numbers. Now Kleene's principal theorem for such functionals is the "first" Recursion Theorem, according to which each such F has a least fixed point (LFP) f, i.e. one that is least among all partial functions g such that $g = F(g)$ (Kleene 1952 p. 348). This is fundamental in the following sense: the partial recursive functions and functionals are just those generated by closing under explicit definition and LFP recursion over the structure $\mathfrak{N}$. For, first of all, one immediately obtains closure under the primitive recursive schemata. Then, given primitive recursive $g(\underline{x}, y)$, one obtains the function $f(\underline{x}) \simeq (\mu y)[g(\underline{x}, y) = 0]$ by taking $f(\underline{x}) \simeq h(\underline{x}, 0)$ where $h(\underline{x}, z) \simeq z$ if $(\forall y < z)\,[g(\underline{x}, y) > 0$

Weihrauch (2000); that features surjective representations from a subset of $\mathbb{N}^{\mathbb{N}}$ to $\mathbb{R}$. Bauer (2000) introduced a still more general theory of representations via a notion of realizability, that allows one to consider classical structures and effective structures of various kinds (including those provided by domain theory) under a single framework; cf. also Bauer and Blanck (2010). The work of Pour-El surveyed in her article (1999) contains interesting applications of the effective approximation approach to questions of computability in physical theory.

[16] Cf. also Blum (2004), to which Braverman and Cook (2006) responds more directly. Actually, the treatment of a number of examples from numerical analysis in terms of the BSS model that takes up Part II of Blum et al. (1997) via the concept of the "condition number" of a procedure in a way brings it in closer contact with the effective approximation model. As succinctly explained to me in a personal communication from Lenore Blum, "[r]oughly, 'condition' connects the BSS/BCSS theory with the discrete theory of computation/complexity in the following way: The 'condition' of a problem instance measures how outputs will vary under perturbations of the input (think of the condition as a normed derivative)." The informative article, Blum (2013), traces the idea of the condition number back to a paper by Turing (1948) on rounding-off errors in matrix computations from where it became a basic common concept in various guises in numerical analysis. (An expanded version of Blum (2013) is forthcoming.) It may be that the puzzle of how the algebraic BSS model serves to provide a foundation for the mathematics of the continuous, at least as it appears in numerical analysis, is resolved by noting that the verification of the algorithms it employs requires in each case specific use of properties of the reals and complex numbers telling which such are "well-conditioned."

$\wedge\ g(\underline{x}, z) = 0]$, else $h(\underline{x}, z')$. It follows that all partial recursive functions (and thence all partial recursive functionals) are obtained by Kleene's Normal Form Theorem.

This now leads one to consider generation of partial functions and functionals by explicit definition and LFP recursion over arbitrary abstract many-sorted structures $\mathfrak{A}$. The development of that idea originates with Platek (1966), a PhD thesis at Stanford that, though never published, came to be widely known by workers in the field. The only requirement on a type-2 functional F on partial functions over $\mathfrak{A}$ for it to have a least fixed point is that F be *monotonic increasing*. In addition, for a thorough-going theory of partial recursive functions and functionals over $\mathfrak{A}$, one must use only those F that have themselves been obtained by LFP recursion in terms of previously defined functions and functionals. For that purpose Platek made use of a hierarchy of hereditarily monotonic 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 at any level in that hierarchy. On the other hand, Platek showed that in the special case that the initial functionals are of type level ≤ 2, everything of type level ≤ 2 that can be generated via explicit definition and LFP recursion in higher types from that data can already be generated via explicit definition and LFP recursion at type level equal to 2. Platek's approach using the full hierarchy of hereditarily monotonic functionals allowed him to subsume and simplify Kleene's theory of recursion in finite types using hereditarily total functionals as the arguments of partial functionals defined by certain schemata (Kleene 1959).[17] Later, Kechris and Moschovakis (1977) showed how to subsume Kleene's theory under LFP recursion at type level ≤ 2 by treating the finite type structure as a many-sorted first-order structure (with infinitely many sorts).

Moschovakis (1984, 1989) took the LFP approach restricted to functionals of type level ≤ 2 in his explanation of the notion of algorithm over arbitrary structures featuring simultaneous LFP definitions, though those can be eliminated in favor of successive LFP definitions of the above form. Both the Platek and Moschovakis approaches are *extensional*. In order to tie that up both with computation over abstract data types and

[17] Platek (1966) also used the LFP approach to subsume recursion theory on the ordinals under the theory of recursion in the Sup functional.

with Bishop's approach to constructive mathematics, in a pair of papers Feferman (1992, 1992a), I extended the use of LFP schemata to cover *intensional* situations, by requiring each basic domain A_i of $\mathfrak{A}$ is to be equipped with an equivalence relation $=_i$ that the initial functions and functionals preserve. The resulting partial functions and functionals are called there *Abstract Computation Procedures* (ACPs). They are successively generated over any structure $\mathfrak{A} = (A_0, A_1, \ldots, A_k, F_0, \ldots, F_m)$ by explicit definition and LFP recursion, where each A_i is non-empty and the F_js are constants, partial functions or monotonic increasing partial functionals of type level 2 over the A_i. Also, one of the domains, say A_0, is fixed to be the booleans $\{t, f\}$ and the operations of negation and conjunction are taken among the basic operations; this allows conditional definition. The details of the schemata may be found in Feferman (2013).

Let us note two comparisons of ACPs with other approaches. First is the following result due to Xu and Zucker (2005): if $\mathfrak{A}$ is an $\mathfrak{N}$-standard structure with arrays, then **While***($\mathfrak{A}$) = **ACP**($\mathfrak{A}$). Secondly, we have a matchup with the Moschovakis (1984) theory of recursors by the result of Feferman (1992a) sec. 9 that the ACPs are closed under simultaneous LFP recursion. In the particular case of the structure $\mathfrak{N}$ of natural numbers, the arguments above in connection with Kleene's First Recursion Theorem show that the partial functions and functionals generated by the abstract computation procedures are just those that are partial recursive. This shows that the effective approximation approach to computation on the reals is accounted for at the second-order level under **ACP**($\mathfrak{N}$), while the Xu-Zucker result shows that the BSS model is subsumed at the first-order level under **ACP**($\mathfrak{R}$).

Clearly it is apt to use the word 'abstract' in referring to the procedures in question since they are preserved under isomorphism. But given the arguments I have made in the preceding sections, it was a real mistake on my part to use 'computation' as part of their designation, and I very much regret doing so. A better choice would have been simply to call them *Abstract Recursion Procedures*, and I have decided to take this occasion to use 'ARP' as an abbreviation for these, in place of 'ACP', thus **ARP**($\mathfrak{A}$) in place of **ACP**($\mathfrak{A}$). The main point now is to bring matters to a conclusion by using these

to propose the following thesis on definition by recursion that in no way invokes the concepts of computation or algorithm.

Recursion Thesis (RT). Any function defined by recursion over a first-order structure $\mathfrak{A}$ (with Booleans) belongs to **ARP**($\mathfrak{A}$).

This presumes an informal notion of being a function f defined by recursion over a first-order structure $\mathfrak{A}$ that is assumed to include the Boolean constants and basic operations. Roughly speaking, the idea for such a definition is that f is determined by an equation $f(x) \simeq E(f, x)$.where E is an expression that may contain a symbol for f and symbols for the initial functions and constants of $\mathfrak{A}$ as well as for functions previously defined by recursion over $\mathfrak{A}$. Now here is the way such a justification for **RT** might be argued. At any given $x = x_0$, $f(x)$ may not be defined by E, for example if $E(f, x)$ = [if $x = x_0$ and f(x) = 0 then 1, else 0]. But if $f(x)$ is defined at all by E, it is by use made of values $f(y)$ that are previously defined. Write $y \prec x$ if $f(y)$ is previously defined and its value is used in the evaluation of x; then let f_x be f restricted to $\{y : y \prec x\}$. Thus the evaluation of $f(x)$ is determined by f_x when it is defined, i.e. $f(x) = E(f_x, x)$ for each such x. It may be that $\{y : y \prec x\}$ is empty if $f(x)$ is defined outright in terms of previous functions; in that case x is minimal in the $\prec$ relation. In the case it is not empty, we may make a similar argument for $f(y)$ for each $y \prec x$, and so on. In order for this to terminate, the $\prec$ relation must be well-founded. Next, take F to be the functional given by $F(f, x) = E(f_x, x)$; F is monotonic increasing, because if $f \subseteq g$ then $f_x = g_x$. So F has a LFP g. But F defines our function f by transfinite recursion on $\prec$, so f is a fixed point of F and hence $g \subseteq f$. To conclude that $f \subseteq g$, we argue by transfinite recursion on $\prec$: for a given x, if $f(y) = g(y)$ for all $y \prec x$ then $f(x) = F(f, x) = E(f_x, x) = E(g_x, x) = F(g, x) = g(x)$. Thus f is given by LFP recursion in terms of previously obtained functions in **ARP**($\mathfrak{A}$) and hence itself belongs to **ARP**($\mathfrak{A}$).

The reason for restricting to first-order structures $\mathfrak{A}$ in the formulation of **RT** is so as not to presume the property of monotonicity as an essential part of the idea of definition by recursion. I should think that all this can be elaborated, perhaps in an

axiomatic form, but if there is to be any thesis at all for definition by recursion over an arbitrary first-order structure (with Booleans), I cannot see that it would differ in any essential way from **RT**. If there is a principled argument for assuming monotonicity of the functionals in a given second-order structure then we would also have a reasonable extension of **RT** to such.

Acknowledgements. I wish to thank Lenore Blum, Andrej Bauer, John W. Dawson, Jr., Nachum Dershowitz, Yuri Gurevich, Grigori Mints, Dana Scott, Wilfried Sieg, Robert Soare, John V. Tucker, and Jeffery Zucker for their helpful comments on an early draft of this article. Also helpful were their pointers to relevant literature, though not all of that extensive material could be accounted for here.

References

Barwise, J. (1975), *Admissible Sets and Structures,* Springer-Verlag, Berlin.

Bauer, A. (2000), *The Realizability Approach to Computable Analysis and Topology*, PhD Thesis, Carnegie Mellon University; Technical Report CMU-CS-00-164.

Bauer, A. and J. Blanck, Canonical effective subalgebras of classical algebras as constructive metric completions, *J. of Universal Computer Science* 16, no. 18, 2496-2522.

Blass, A. and Y. Gurevich (2003), Algorithms: A quest for absolute definitions, in *Bull. EATCS* 81, 195-225.

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

_____ (2013), Alan Turing and the other theory of computation, in Cooper and van Leeuwen (2013), 377-384.

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-Verlag, New York.

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

Church, A. (1936), An unsolvable problem of elementary number theory, *American J. of Mathematics* 58, 345-363.

_____ (1937), Review of Turing (1936-37), *J. Symbolic Logic* 2, 42-43.

Copeland, B. J. (2002), The Church-Turing Thesis, http://plato.stanford.edu/entries/church-turing/, Stanford Encyclopedia of Philosophy.

Cooper, S. B. and J. van Leeuwen, eds. (2013), *Alan Turing: His work and impact*, Elsevier Pub. Co., Amsterdam.

Davis, M. (1982), Why Gödel didn't have Church's thesis, *Information and Control* 54, 3-24.

Dawson, J. W., Jr. (1993), Prelude to recursion theory: the Gödel-Herbrand correspondence, in *First International Symposium on Gödel's Theorems* (Z. W. Wolkowski, ed.), World Scientific, Singapore, 1-13.

Dershowitz, N. and Y. Gurevich (2008), A natural axiomatization of computability and proof of Church's Thesis, *Bull. Symbolic Logic* 14, 299-350.

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

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

_____ (2013) About and around computing over the reals, in *Computability. Turing, Gödel, Church, and beyond* (B. J. Copeland, C. J. Posy, and O. Shagrir, eds.), MIT Press, Cambridge, 55-76.

Fenstad, J. (1980), *General Recursion Theory. An axiomatic approach*, Springer-Verlag, Berlin.

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.

Gandy, R. O. (1980), Church's thesis and principles for mechanisms, in *The Kleene Symposium* (J. Barwise, H. J. Keisler, and K. Kunen, eds.), North-Holland, Amsterdam, 123-145.

_____ (1988), The confluence of ideas in 1936, in *The Universal Turing Machine. A half-century survey* (R. Herken, ed.), Oxford Univ. Press, Oxford, 55-111.

Gödel, K. (1933), The present situation in the foundations of mathematics, (unpublished lecture), in Gödel (1995), 45-53.

_____(1934), On undecidable propositions of formal mathematical systems, (mimeographed lecture notes by S. C. Kleene and J. B. Rosser), reprinted with revisions

in Davis, M. (1965) (ed.), *The Undecidable. Basic papers on Undecidable propositions, unsolvable problems, and computable functions*, Raven Press, Hewlett, NY, 39-74, and in Gödel (1986), 346-371.

_____(1986), *Collected Works. Vol. I, Publications 1929-1936* (S. Feferman et al., eds.), Oxford Univ. Press, New York.

_____ (1990), *Collected Works. Vol. II, Publications 1938-1974* (S. Feferman et al., eds.), Oxford Univ. Press, New York.

_____(1995), *Collected Works. Vol. III, Unpublished Essays and Lectures* (S. Feferman et al., eds.), Oxford Univ. Press, New York.

_____(2003), *Collected Works. Vol. V, Correspondence H-Z* (S. Feferman et al., eds.), Oxford Univ. Press, Oxford.

Griffor, E. (ed.) (1999), *Handbook of Computability Theory*, Elsevier, Amsterdam.

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

Gurevich, Y. (2000), Sequential abstract state machines capture sequential algorithms, *ACM Transactions on Computational Logic* 1, 77-111.

_____ (2012), What is an algorithm?, *SOFSEM 2012: Theory and Practice of Computer Science* (M. Bielikova et al., eds.), LNCS 7147 (2012), 31-42.

Hilbert, D. (1926), Über das Unendliche, *Mathematische Annalen* 95, 161-190; English translation in van Heijenoort (1967) (ed.), *From Frege to Gödel. A source book in mathematical logic, 1879-1931*, Harvard Univ. Press, Cambridge MA, 367-392.

Hinman, P. G. (1999), Recursion on abstract structures, in Griffor (1999), 315-359.

Kechris, A. and Y. Moschovakis (1977), Recursion in higher types, in *Handbook of Mathematical Logic* (J. Barwise, ed.), North-Holland Pub. Co., Amsterdam, 681-737.

Kleene, S. C. (1936), General recursive functions of natural numbers, *Mathematische Annalen* 112, 727-742.

_____(1936a), λ-definability and recursiveness, *Duke Mathematical Journal* 2, 340-353.

_____ (1943), Recursive predicates and quantifiers, *Transactions American Math. Soc.* 53, 41-73.

_____ (1952), *Introduction to Metamathematics*, North-Holland, Amsterdam.

_____ (1959), Recursive functionals and quantifiers of finite type I, *Transactions American Math. Soc.* 91, 1-52.

Kolomogorov, A. N. and V. A. Uspenski (1953), On the definition of an algorithm, *Uspehi Math. Nauk.* 8, 125-176; *American Math. Soc. Translations* 29 (1963), 217-245.

Kreitz, C. and K. Weihrauch (1984), A unified approach to constructive and recursive analysis, in *Computation and Proof Theory* (M. Richter, et al., eds.), *Lecture Notes in Mathematics* 1104, 259-278.

_____ (1985), Theory of representations*, Theoretical Computer Science* 38, 35-53.

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.

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

_____ (1980a), 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.

_____ (2001), What is an algorithm?, in *Mathematics Unlimited—| 2001 and beyond* (B. Engquist and W. Schmid, eds.), Springer, Berlin, 919-936.

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

Pour-El, M. B. (1999), The structure of computability in analysis and physical theory, in Griffor (1999), 449-471.

Rogers, H. (1967), *Theory of Recursive Functions and Effective Computability,* McGraw-Hill Publ. Co., New York, NY.

Sacks, G. E. (1990), *Higher Recursion Theory*, Springer-Verlag, Berlin.

Shagrir, O. (2002), Effective computation by humans and machines, *Minds and Machines* 12, 221–240.

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

Sieg, W. (2002), Calculations by man and machine: Conceptual analysis, in *Reflections on the Foundations of Mathematics. Essays in honor of Solomon Feferman,* (W. Sieg, R.

Sommer, C. Talcott, eds.), *Lecture Notes in Logic* 115, *Assoc. for Symbolic Logic,* A. K. Peters, Ltd., Natick, MA, 390-409.

_____ (2002a) Calculations by man and machine: Mathematical presentation, in *Proceedings of the Cracow International Congress of Logic, Methodology and Philosophy of Science*, Synthese Series, Kluwer Academic Publishers, Dordrecht, 245-260.

_____ (2003), Introductory note to the Gödel-Herbrand correspondence, in Gödel (2003), 3-13.

_____ (2005), Only two letters: The correspondence between Herbrand and Gödel, *Bull. Symbolic Logic* 11, 172-184.

_____ (2008), Church without dogma: Axioms for computability, in *New Computational Paradigms* (B. Löwe, A. Sorbi, B. Cooper, eds.), Springer-Verlag, Berlin, 139-152.

_____ (2013) Axioms for Computability: Do they allow a proof of Church's Thesis?, in *A Computable Universe – Understanding and exploring nature as computation* (H. Zenil, ed.), World Scientific Publishing, Singapore, 99-123.

Soare, R. I. (1996), Computability and recursion, *Bull. Symbolic Logic* 2, 284-321.

_____ (1999), The history and concept of computability, in Griffor (1999), 3-36.

Stoltenberg-Hansen, V. and J. V. Tucker, Computable rings and fields, in Griffor (1999), 363-447.

Tucker, J. V. (1980), Computing in algebraic systems, in *Recursion Theory, its Generalizations and Applications* (F. R. Drake and S. S. Wainer, eds.), Cambridge, Univ. Press, Cambridge.

Tucker, J. V. and J. I. Zucker (1988), *Program Correctness over Abstract Data Types,* CWI Monograph, 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 Univ. Press, Oxford, 317-523.

_____ (2006), Abstract versus concrete computability: The case of countable algebras, in *Logic Colloquium '03* (V. Stoltenberg-Hansen and J. Väänänen, eds.), *Lecture Notes in Logic* 24, Assoc. for Symbolic Logic, A. K. Peters, Ltd., Wellesley, MA, 377-408.

Turing, A. (1936-37), On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Math. Soc. Ser. 2,* 42, 230-265; a correction, *ibid.* 43, 544-546.

_____ (1937), Computability and λ-definability, *J. Symbolic Logic* 2, 153-163.

_____ (1948), Rounding-off errors in matrix processes, *Quart. J. Mech. Appl. Math.*1, 287-308; reprinted in Cooper and van Leeuwen 2013, 385-402.

Weihrauch, K. (2000), *Computable Analysis*, Springer, New York.

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

Dept. of Mathematics

Stanford University

Email: feferman@stanford.edu