FINDING THE BEST MODEL FOR CONTINUOUS COMPUTATION

REBECCA ABIGAIL RESNICK

ABSTRACT. While the theory of computability over countable sets is well-defined and flexible, the definition of computability over *continuous* sets (e.g. the real numbers), without the fortification provided by the Church-Turing Thesis, is much more contentious. Since Turing's introduction of a universal device for computation over countable sets (the "universal Turing Machine"), several demonstrably non-equivalent formalizations of the intuitive notion of continuous (alternately: analog) computation, and more specifically, computation over the real numbers, have been proposed. None of these is yet accepted by the majority of mathematicians, and, as a result, the contemporary landscape of research into continuous computation is factional. I will present and compare several of dominant theories of continuous computation, including techniques from recursive analysis and the Blum-Smale-Shub model.

1. INTRODUCTION

Continuous computation tries to answer the question of how to define the concept of computability in non-discrete situations. In contrast to the discrete case, in which the Church-Turing thesis and related, provably equivalent models[1] provide fortification for our intuition with regards to the definition of computability, the intuition for what it means to compute a quantity or set of values in the continuous case is much less well-defined. To date, no single model of continuous computation has been accepted as canon in the same manner as Turing's model for discrete computability. Moreover, several demonstrably non-equivalent models have been proposed [BSS89, Moo95, Koi97, BH06].

In this paper, I will discuss two prominent, non-equivalent models of continuous computation and related variations: recursive (or computable) analysis, described initially by Gryzegorczyk [Grz55, Grz57] and Lacombe[2], and the Blum-Smale-Shub (BSS) model [BSS89]. The basic attributes of these models are described very briefly below:

(1) **Recursive analysis**. This model of continuous computation is based on the definition of computable functionals as introduced by Grzegorczyk [Grz55, Grz57]. Recursive analysis gets at the problem of computing on infinitely long decimal representations (i.e. real numbers) by developing

Submitted on March 21, 2011. The author is an undergraduate senior honors candidate in Mathematics at Harvard University.

[1]see [LS01] in particular for a description of Diophantine equations as an equivalent model of classical computation.

[2]The complete citation information for Lacombe's article is:

D. Lacombe, Extension de la notion de fonction rcursive aux fonctions dune ou plusieurs variables relles, *C. R. Acad. Sci. Paris*, 240 (1955), 2478-2480; 241 (1955), 13-14, 151-153.

I was unable to find a copy of Lacombe's article. Luckily, as we shall see in section 3.2, Grzegorczyk describes Lacombes definition of real computability in order to prove it equivalent to his own.

a system of increasingly accurate approximations [PR89, Wei95, Wei97]. This model has applications in analysis, and topology, and physics [PR89, Moo96, BH06].

(2) The **Blum-Smale-Shub (BSS)** model [BSS89], and related variations [Koi97, Bra97, Zho98]. The BSS model assumes the compression of infinite-length operations into finite time and seeks to algebratize the theory of continuous computation to the greatest degree possible, along the lines of the algebraic circuits of computational complexity theory [AB09]. Unsurprisingly, this model has potential and demonstrated ramifications within computational complexity theory [BSS89, BCSS97, BC06].

The non-equivalence of these models stems from a fundamental disagreement over exactly what continuous computation means, and what a model of it should represent. Recursive analysis, which defines calculation over the real numbers as an infinite sequence of classical calculations over the rationals (subject to uniformizing and effectivizing constraints–see section 3), is a model of the way modern (non-quantum) computers actually work. However, even from the perspective of computer science, this is not the whole story. As we shall see in section 4, it is often fruitful, when studying more theoretical aspects of computation, to make the assumption of infinite precision, as in the BSS model.

The question may arise at this point in the mind of the reader of why continuous computation is a fruitful area of research. The study of the theory of computation is an attempt to formalize the rules that govern the way that computers, including theoretical representations of computational devices, work. Yet one might argue that the devices described within the study of continuous computation bear little relation to anyone's conception of reality. The very idea of continuous computability, which assumes the existence of infinitely precise devices or at least infinitary sequences of rational approximations, is a perversion of the viscerally intuitive notion of classical computability, as formulated by Turing [Tur36] into a concept which bears little relation to either the way that physical computers operate or the theoretical way in which we think about calculating any given quantity.

To this theoretical objection, I offer two counterarguments, one practical and one philosophical. As a practical consideration, as previously mentioned, the objects defined within the study of continuous computation, while abstract in their own right, have proved quite useful in areas of research outside of the realm of recursion theory. The Blum-Smale Shub (BSS) model [BSS89] in particular has lead to recent developments in the field computational complexity theory, [BCSS97, BC06, AB09], while the techniques of recursive analysis have been influential in analysis and physics [PR89]. The BSS model and its applications will be presented in section 4 and recursive analysis will be explored in section 3. Even Moore's model, [Moo95], a younger and less-explored model of continuous computation based on functional integration has demonstrated relations to descriptive set theory and potential applications within the study of neural nets [Moo95].

On the philosophical side, it is worth noting that, although the contemporary tools of continuous computation are physically unrealizable and, especially in the case of the models of Blum-Smale-Shub and Moore, not easily approximated by physically constructible objects[3], the questions at the the center of the study of

[3]In contrast, Turing machines, while also physically untenable due to the infinite capacity of their computing space, are easily approximated by physical machines; for example: computers.

continuous computation are extremely intuitive. The exact calculation of irrational numbers, an archetypal problem within continuous computation, for example, is impossible within the framework of classical computation. The fact that such numbers exist and that we can conceive of exact representations of them is itself a justification for a finer filter for our definition of what is and is not computable.

As a secondary example of the inherence of continuous computation within the mathematical universe, consider Steffensen's method for calculating roots of real-valued polynomials [DB03]. Without at least an intuitive notion of what it means to compute on a real-valued function, this technique is meaningless. I will discuss the decidability of Steffensen's method with respect to continuous computation in greater detail in section 4 on the Blum-Smale-Shub model.

2. Background

Before entering into the murky waters that characterize contemporary research in *continuous* computation, we make a brief introduction to the more fundamental concept of computability itself. Turing gives a general definition of the nature of computability as follows :

Definition 2.1. [Tur36] A number is **computable** if its decimal representation can be calculated by finite means.

This statement is an appeal to intuition rather than a definition of terms. Turing expects that the educated reader has a sufficiently standardized, if definitionally nebulous, notion of what it means to calculate a quantity in using finite resources. The shared nature of this fundamental intuition is, indeed, the crux of Turing's argument. By leaving the terms of his definition assertively vague, Turing opens the door to all manners of models for computation. His argument, one half of the famous Church-Turing thesis, is, roughly, that all "reasonable" models of computation are equivalent. [4]

Turing goes on to specify several models for discrete computation and to prove their equivalence. For the purposes of this exposition, I will use Turing machines (Turing himself called them *a*- or *automatic*-machines [Tur36]), described briefly below, as a model for discrete computation unless otherwise specified.

A Turing machine is, in the most general sense, an algorithm for describing a computation. Turing machines model individual computations over a finite set of symbols (the "alphabet") as a series of steps which are recorded on an infinitely long workspace (the "tape"). A given Turing machine may have only finitely many configurations (called "states"), and in any step in the calculation, the only actions that the Turing machine may make are to move one step in either direction on the tape and/or to write or delete something from the the position on the tape that is currently occupied. Formally:

[4]The Church-Turing thesis, which owes its moniker to Stephen Kleene [Kle52], is so-named because it combines Church's argument that the λ-calculus is equivalent to recursive function theory [Dav65] with Turing's argument that that his invention, the Turing machine is equivalent to recursive function theory [Tur36]. The Church-Turing thesis is, of course, unprovable in the general case. However, by demonstrating the equivalence of three foremost models of computation–recursion, λ-calculus and Turing machines–Church's and Turing's arguments together provide compelling evidence that any reasonable model of computation should, at the very least, agree with Church's and Turing's ideas.

Definition 2.2. [Sip05] A **Turing machine** is a 7-tuple $(S, \Sigma, \Gamma, \delta, s_0, s_a, s_r)$ abiding by the the following criteria:

(1) S, Σ, Γ are finite sets,
(2) S is the set of states,
(3) Σ is the input alphabet, with a "blank symbol", $\sqcup \notin \Sigma$. At the beginning of any calculation, all except for finitely many spaces on the tape are, by default, populated by the blank symbol.
(4) Γ is the tape alphabet, with $\Sigma \cup \{\sqcup\} \subseteq \Gamma$. We assume that, at the beginning of any computation, the Turing machine has only the input written on the tape.
(5) $\delta : S \times \Gamma \to S \times \Gamma \times \{L, R\}$ is the transition function,
(6) $s_0 \in \Sigma$ is the start state,
(7) $s_a \in \Sigma$ is the accept state,
(8) and $s_r \in \Sigma$ is the reject state.

For a more complete discussion of Turing machines and an introduction to discrete computation, see [Sip05].

A number in $\mathbb{R}$ is then called "computable" in Turing's sense if there is some Turing machine that accepts, in finite time, only its decimal representation. That is,

Definition 2.3. A number $x \in \mathbb{R}$ is **computable** if there exists some Turing machine T_x such that

$$T_x(y) = \begin{cases} 1 & : y = x \\ 0 & : \text{otherwise.} \end{cases}$$

It is important to note that, for a number to be computable, the Turing machine must decide whether to accept (return 1) or reject (return 0) on *all* inputs. Note that the definition for a Turing machine does not prescribe this to be the case in general. Nothing, a priori, precludes an arbitrary Turing machine from entering an infinite loop of calculation on a given input and never returning an answer. Turing machines that *do* reach an answer (or, to use the accepted parlance, **halt**) on all inputs are called **computable**, or, synonymously, **decidable** or **recursive**. Turing machines that do not have this very limiting property are called **partial computable** or **partial recursive**.

This definition implies automatically that all rational numbers are computable for any rational $q \in \mathbb{Q}$ with decimal representation $q_0 q_1 q_2 q_3 \dots q_n$ ($q_k =$ "." for some $k \leq n, q_i \in \{0, 1, \dots, 9\}$ for all other i). Let T_x be a Turing machine over the alphabet $\Sigma = \{0, 1, \dots, 9, \text{"."}\}$, with $k + 3$ states, whose transition map δ is given by the finite graph in figure 1.

Assume, at the outset of the computation, T_x's working tape contains the input in its first k' spaces and is blank everywhere else. T_x reaches the accept state precisely when the input is the decimal representation of x and reaches the reject states in all other cases. This concept of reaching a decision (accept or reject) on a given input leads to a definition of computability for sets:

Definition 2.4. A set $A \subseteq \mathbb{N}$ is **computable** if there exists a Turing machine T_A such that

$$T_A(n) = \begin{cases} 1 & : n \in A \\ 0 & : \text{otherwise.} \end{cases}$$

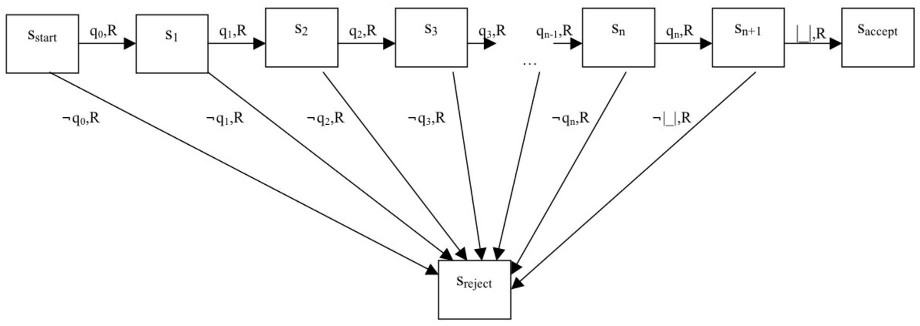

FIGURE 1. A Turing machine which accepts only the rational number $q = q_0 q_1 \ldots q_n$ and halts in the reject state on all other inputs.

Considering functions from $\mathbb{N} \to \mathbb{N}$ as subsets of $\mathbb{N} \times \mathbb{N}$ (members of which themselves can be coded as single natural numbers[5]), the definition of computability for sets thus gives rise naturally to a definition of functional computability:

Definition 2.5. A function $f : \mathbb{N} \to \mathbb{N}$ is **computable**[6] if there exists a Turing machine T_f such that

$$T_f(n, m) = \begin{cases} 1 & : m = f(n) \\ 0 & : \text{otherwise.} \end{cases}$$

While flexible, this definition of functional computability is quite limited. All continuous functions and any function whose domain or range is non-discrete is immediately disqualified by the inescapable presence of non-repeating decimals. It is the narrowness of this definition of computability that drives the study of continuous computation, which is alternately and illustratively called recursion over the reals. In contrast to Turing's insistence that the concept computability be grounded in finite calculations, the study of *continuous* computation asks what happens if we allow certain of our "means" of calculation to be infinite and even uncountable.

3. Recursive Analysis

3.1. **Background.** In the search for a coherent definition of continuous computation, recursive analysis has an initial leg up in terms of understanding because it relies, to the greatest degree of any of the models presented in this paper, on the terminology and notation established within Turing's model of classical computation. In the most general sense, recursive analysis tries to address the computational difficulty of calculating on non-repeating decimals by using a sequence of computable Turing machine-like devices to get an arbitrarily close approximation.

[5][Mos09] Let $p(i) = p_i$ be the ith prime number. Then a finite tuple of numbers, $(t_0, \ldots, t_n)$ is uniquely coded by the number

$$\langle\langle t_0, \ldots_n \rangle\rangle = p_0^{t_0+1} \cdot \cdots \cdot p_n^{t_n+1}.$$

[6]This definition is a slight alteration on Sipser, who states that a function $f : \mathbb{N} \to \mathbb{N}$ is computable if there exists a Turing machine T_f such that $T_f(n)$ finishes its calculation with only $f(n)$ written on the tape.

The intuitive foundation of recursive analysis is the conceptualization of real numbers as limits of Cauchy sequences of rationals.[7] With this formulation in mind, recursive analysis views a computation over the reals as a sequence of increasingly accurate Turing-machine computations over the rationals. Recursive analysis puts several significant restraints on the sequence of Turing machines used for a given computation, most notably that each Turing machine is totally computable (recursive) and that the specifics of the entire sequence are *themselves* computable from the start. That is, we know exactly what the nth Turing machine in the sequence will look like, even though we may not know the result of its computation for some time. Formally,

Definition 3.1. [PR89] A sequence $\langle r_k \rangle_{k\in\mathbb{N}}$ with $r_k \in \mathbb{Q}$ is **computable** if there exist computable functions $a, b, s : \mathbb{N} \to \mathbb{N}$, such that for all k, $b(k) \neq 0$ and

$$r_k = (-1)^{s(k)} \frac{a(k)}{b(k)}.$$

This equation provides a coding of any rational number as a 3-tuple of integers (a, b, c), and allows us to code any sequence of rational numbers within three *computable* sequences of integers $\langle a(k), b(k), c(k) \rangle_{k\in\mathbb{N}}$. As with Cauchy sequences, we wish to use these countable sequence $\langle r_k \rangle$ of rationals to represent arbitrarily close approximations of real numbers, so we must define a notion of the effectiveness of such sequences in converging to real numbers:

Definition 3.2. [PR89] Given $x \in \mathbb{R}$, a sequence $\langle r_k \rangle_{k\in\mathbb{N}}$ with $r_k \in \mathbb{Q}$ **converges effectively** to x if there exists a computable function $e : \mathbb{N} \to \mathbb{N}$ such that for all $n \in \mathbb{N}$, we have

$$k \geq e(n) \implies |r_k - x \leq 2^{-n}|.$$

This definition assures that if we wait long enough, our approximation will *always* be within a certain distance of x and that this distance approaches 0 as $n \to \infty$. The "all" in this definition is important because it assures that our approximations can only get better in later steps: once we enter a certain confidence interval, we never leave it again. In the obvious way, we then define a real number x to be a **computable real** (or simply "computable" if there is no ambiguity) if there exists a computable sequence of rationals that converges effectively to x.

It is a jarring but relatively easy to prove fact that, defined in this manner, not all computable real numbers can be compared effectively. More specifically, we can devise a scenario in which we are be unable to determine whether or not two numbers are equal. Let $\mathbb{Q}^* = \mathbb{Q} - \{0\}$. We wish to show:

Fact 3.3. [PR89] *Given a computable real number x, there is an effective procedure for showing $x \in \mathbb{Q}^*$ if and only if $x \neq 0$*

This fact seems just short of cyclic, but it speaks to the problems of providing effective proofs using approximations, even very close ones, for real numbers when 0 is involved. The basic idea of the proof is that, for any real number $x \in \mathbb{Q}^*$, we can eventually bound x effectively away from 0. We provide a counterexample to show that this is not the case for $x = 0$.

[7]This method of defining real numbers is the starting point for Pour-El and Richard's [PR89] discussion on Recursive Analysis. For a discussion of the construction of the reals as Cauchy sequences of the rationals, see [KF75].

Proof. We follow the proof and example given in [PR89]. Suppose $x > 0$ is a computable real and let $\langle r_k \rangle_{k\in\mathbb{N}}$ be a computable sequence that converges effectively to x. Let $N = 0$ and consider the following procedure:

(1) Compute $e(N)$ and $r_{e(N)}$.
(2) If $r_{e(N)} > 2^{-N}$, return "true."
(3) $N++$, return to step 1.

Since $x > 0$, this process must eventually terminate. Indeed, take N such that $2^{-N} < x/2$. Then since we have $|r_{e(N)} - x| \leq 2^{-N}$, a quick triangle inequality gives us $r_{e(N)} > 2^{-N}$.

The proof for $x < 0$ is analogous. Now consider $x = 0$. We show via counterexample that there is not necessarily an effective procedure for demonstrating that this is the case. To do so, we first note that a **double sequence** $\langle x_{n,k} \rangle_{n,k\in\mathbb{N}}$ can be mapped onto a single-index sequence $\langle x_j \rangle_{j\in\mathbb{N}}$ using the standard recursive mapping from $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ [8], and thus, can be thought of as computable under the same definition. We will also make use of the following fact:

Fact 3.4. (Closure Under Effective Convergence) [PR89] *Let $\langle x_{n,k} \rangle$ be a computable double sequence of real numbers such that $x_{n,k}$ converges effectively to some number x_n for each n as $k \to \infty$. Then $\langle x_n \rangle$ is computable.*

The proof of this fact is straightforward (and not terribly interesting), so we omit it.

Consider $\langle x_{n,k} \rangle_{n,k\in\mathbb{N}}$ given by [PR89]:

$$x_{n,k} = \begin{cases} 2^{-m} & : \text{if } n = a(m) \text{ for some } m \leq k, \\ 0 & : \text{otherwise.} \end{cases}$$

Where $a : \mathbb{N} \to \mathbb{N}$ is an injective, recursive function which generates a recursively enumerable, *non*-recursive set A. Then $x_{n,k} \to x_n$ as $k \to \infty$ where x_n is given by the following:

$$x_n = \begin{cases} 2^{-m} & : \text{if } n = a(m) \text{ for some } m, \\ 0 & : \text{otherwise.} \end{cases}$$

Whereas for each $x_{n,k}$, we had to check only finitely many possible m's to determine $x_{n,k}$, for x_n we are not so lucky. For each n, in order to determine whether $x_n = 0$, we need to know whether there exists *any* m among all natural numbers such that $a(m) = n$. While a is recursive by definition, the successive calculation of $a(m)$ for (potentially) all m is not necessarily finite.

If we could show $\langle x_n \rangle$ to be a computable sequence of real numbers as per definition 3.1, then we would be done. For, as we have just described, we would need to perform an infinite number of calculations in order to determine whether $x_n = 0$ for any n. Thus, there is no way to bound x_n effectively at 0, even though it might be the case that $x_n = 0$.

In order to show $\langle x_n \rangle$ to be a computable sequence of real numbers, note that $x_{n,k} \neq x_n$ only when there exists m such that $a(m) = n$, and the least such $m > k$. Then we have $x_{n,k} = 0$, and, by the way we define x_n, we have $x_n = 2^{-m} < 2^{-}k$. Thus, for all k, n,

[8] See footnote 5.

$$|x_{n,k} - x_n| < 2^{-k}.$$

By fact 3.4, this implies that $\langle x_n \rangle$ is computable. Thus, x_n is the number that we sought. □

Proponents of the Recursive Analysis model are untroubled by this inherent uncertainty in comparing real numbers. Like the close cousins, physics and classical analysis, from which the techniques of Recursive Analysis stem, this model of continuous computation relies heavily on approximation when exactitude is out of reach. Pour-El and Richards [PR89] note that, while comparisons are not necessarily possible for computable reals, they are possible for the computable sequence of rationals from which computable reals are formed. Setting

$$r_n = (-1)^{s(n)} \frac{a(n)}{b(n)},$$

we have an effective test for picking out the sequences $\{n : r_n = 0\}, \{n : r_n > 0\}$ and $\{n : r_n < 0\}$. Namely

(1) $r_n = 0 \leftrightarrow a(n) = 0$,
(2) $r_n > 0 \leftrightarrow a(n) > 0$ and $s(n)$ is even,
(3) $r_n < 0$ otherwise.

This ideology, that approximations are inevitable, even in comparisons of infinite numbers, is the sharpest contrast between the Recursive Analysis model, and the models of Moore and Blum, Smale and Shub. Whereas recursive analysis founds its formulation on the acceptance of quantities that are incomparable, both the Moore model and BSS make the drastic and simplifying assumption that infinite calculations are achievable in finite time.[9]

3.2. **Recursive Function(al)s.** Recursive analysis developed out of a need for precision in defining computability on continuous sets, especially within the continuous scenarios presented by physics and analysis. Many of scenarios are played out with functions over $\mathbb{R}^n$ or $\mathbb{C}^n$ (or related subsets) and, with the definition of computation for real numbers in hand, we may move on to defining a theory of computability for functions over and into such continuous sets. Grzegorczyk [Grz55, Grz57] gives the first formulation of the theory that would eventually grow into recursive analysis. He defines computability for real functions in terms of **functionals**, here defined in the most general sense to be functions of functions. Grzegorczyk conceives of real numbers as functions

$$a_r : \mathbb{N} \to \{0, 1, \ldots, 9\},$$

where $a_r(n)$ codes the nth digit of the real number r. Continuous functions are thus defined to be functionals over the functions describing the real numbers. Let $\mu u S(u)$ be defined in the usual way to mean the "least u such that the statement $S(u)$ is true." Using Grzegorczyk's notation, let the set $\mathfrak{N}$ be given by

$$\mathfrak{N} = \{a : \mathbb{N} \to \mathbb{N}\}.$$

So $\mathfrak{N}$ is exactly the set of functions describing the real numbers. Computability for continuous functions is defined in terms of computability for functionals:

[9]This is not necessarily a philosophically or mathematically untenable assumption. See [Bar05].

Definition 3.5. [Grz55] The **class** of **computable functionals**, $\mathfrak{K}$, is defined to be the least class that abides by the following rules (where all $a \in \mathfrak{N}$)

(1) $\mathfrak{K}$ contains the following initial functions:
  (a) The identity functional:
$$\mathbf{I}\left\langle a\right\rangle(x) := a(x)$$
  (b) The functionals:
$$-\left\langle a\right\rangle(x,y) := -(x,y) := x\dot{-}y := \begin{cases} x-y & : \text{if } x>y \\ 0 & : \text{otherwise.}\end{cases}$$
$$\mathbf{E}\left\langle a\right\rangle(x,y) := E(x,y) := x^y$$
$$\mathbf{S}\left\langle a\right\rangle(x,y) := S(x,y) := x+1.$$
  So we automatically start with a way to find successors and (positive) differences between numbers and to exponentiate.

(2) $\mathfrak{K}$ is closed under the following operations:
  (a) Substitution, defined in the usual way.
  (b) Identification of variables representing numbers with those representing functions. Specifically, if $\alpha \in \mathfrak{K}$, where α takes n variables as arguments, and
$$\beta\left\langle a\right\rangle(x_1,\dots,x_{k-1}) = \alpha\left\langle a\right\rangle(x_1,\dots,x_j,\dots x_{k-1},x_j),$$
$$\gamma\left\langle a_1,\dots,a_{n-1}\right\rangle(x_1,\dots,x_k) = \alpha\left\langle a_1,\dots,a_j,\dots,a_{n-1},a_j\right\rangle(x_1,\dots,x_k).$$
  Then $\beta,\gamma \in \mathfrak{K}$.
  (c) Effective minimum: i.e. if $\alpha \in \mathfrak{K}$, where α takes n functions and k variables as arguments, and there exist $a_1,\dots,a_n$ and $x_1,\dots,x_{k-1}$ such that
$$\sum_{u\in\mathbb{N}} \alpha\left\langle a_1,\dots,a_k\right\rangle(u,x_1,\dots,x_{k-1}) = 0,$$ [10]
  then β defined in the following way:
$$\beta\left\langle a_1,\dots,a_n\right\rangle(x_1,\dots,x_{k-1}) = (\mu u)[\alpha\left\langle a_1,\dots,a_n\right\rangle(u,x_1,\dots,x_{k-1}) = 0]$$
  is also a member of $\mathfrak{K}$.

The set of *continuous* computable functions, $\mathfrak{K}_1$, is thus defined by the standard definition for continuity, modified to fit within the bounds of computable functional theory as follows:

Definition 3.6. [Grz57] A function $f : \mathbb{R} \to \mathbb{R}$ is **computable continuous** (i.e. $f \in \mathfrak{K}_1$) if and only if there exists a functional $\alpha \in \mathfrak{K}$ such that for all $r \in \mathbb{R}$ and all $a \in \mathfrak{N}$, we have
$$\forall k \in \mathbb{N}\left[\left|\frac{a(k)}{k+1} - r\right| < \frac{1}{k+1}\right] \implies \forall k \in \mathbb{N}\left[\left|\frac{\alpha\left\langle a\right\rangle(k)}{k+1} - f(r)\right| < \frac{1}{k+1}\right].$$

[10]Grzegorczyk describes this condition in a more set theoretic manner as
$$\prod_{a_1,\dots,a_n\in\mathfrak{N}}\ \prod_{x_1,\dots,x_{k-1}\in\mathbb{N}}\ \sum_{u\in\mathbb{N}} \alpha\left\langle a_1,\dots,a_k\right\rangle(u,x_1,\dots,x_{k-1}) = 0.$$

Grzegorczyk's definition is robust because, as he proves in [Grz57], his conception of what constitutes the set of continuous computable functions is equivalent to several earlier definitions. To simplify future notation, let $\langle q_n \rangle_{n\in\mathbb{N}}$ be a recursive enumeration without repetition of all rational numbers and let $\langle s_n \rangle_{n\in\mathbb{N}}$ be a computable sequence of all open rational segments. $\langle q_n \rangle$ is clearly classically computable and $\langle s_n \rangle$ is computable in the sense of [PR89]. Then the results of [Grz57] are summarized as follows:

Theorem 3.7. [Grz57] *The following are equivalent:*

(1) $\mathfrak{K}_1$,
(2) $\mathfrak{K}_2$ *given by the following:* $f \in \mathfrak{K}_2$ *if and only if:*[11]
  (a) *for any computable sequence* $\langle r_k \rangle$*, the sequence* $\langle f(r_k) \rangle$ *is also computable.*
  (b) *There exists a (classically) computable integral function* g *such that for all* $m, n, k \in \mathbb{N}$ *and* $a, b \in \mathbb{R}$*,*

$$(q_n < a, b < q_m) \wedge \left(|a-b| < \frac{1}{g(n,m,k)} \right) \implies |f(a) - f(b)| < \frac{1}{k+1}.$$

  In other words, f *is computably uniformly continuous within the rational segments.*
(3) $\mathfrak{K}_3$ *given by the following:* $f \in \mathfrak{K}_3$ *if and only if*
  (a) f *is continuous,*
  (b) $\langle f(q_n) \rangle$ *is continuous,*
  (c) $\langle f(q_n) \rangle$ *is computably uniformly continuous within the rational segments.*
(4) $\mathfrak{K}_4$ *given by the following:* $f \in \mathfrak{K}_4$ *if and only if:*[12]
  (a) $a \in s_n \implies f(a) \in s_{f(n)}$,
  (b) $f(a) \in s_m \implies \exists n \in \mathbb{N}[(a \in s_n) \wedge (s_{f(n)} \subseteq s_m)]$,
  (c) $(s_n \subseteq s_k) \wedge (n \geq k) \implies s_{f(n)} \subseteq s_{f(k)}$.

Proof. Grzegorczyk demonstrates the necessary inclusions by introducing three auxiliary sets, $\mathfrak{K}_2', \mathfrak{K}_3'$, and $\mathfrak{K}_4'$ given as follows:

A function $f \in \mathfrak{K}_2'$ if and only if:

(1) $\langle f(q_n) \rangle$ is computable,
(2) f is computably uniformly continuous.

A function $f \in \mathfrak{K}_3'$ if and only if:

(1) f is continuous,
(2) there exist computable, integer-valued functions, α, β such that for all $k, l, m, n, t \in \mathbb{N}$,

$$\left| \frac{\alpha(n,k)}{k+1} - f(q_n) \right| < \frac{1}{k+1},$$

$$(r_n < r_l) \wedge (r_t < r_m) \wedge \left(|r_l - r_t| < \frac{1}{\beta(n,m,k)} \right) \implies |\alpha(l,k) - \alpha(t,k)| < 3.$$

[11]Grzegorczyk cites the definition of $\mathfrak{K}_2$ to an unpublished paper of Mazur entitled *Introduction to the computable analysis.* In contemporary parlance, this definition of real computability is known as Banach/Mazur computability [Wei00]

[12]This is Lacombe's definition.

A function $f \in \mathfrak{K}'_4$ if and only if there exists a functions computable, integer-valued function α:

(1) $a \in s_n \implies f(a) \in s_{\alpha(n)}$
(2) $b \neq f(a) \implies \exists n \in \mathbb{N}(a \in s_n) \wedge b \notin s_{\alpha(n)}$.

The $\mathfrak{K}'_i$'s serve as a bridge between the $\mathfrak{K}_i$'s and the $\mathfrak{K}_{i+1}$'s. Grzegorczyk shows the following sequences of inequalities:

$$\mathfrak{K}_1 \subset \mathfrak{K}_2 \subset \mathfrak{K}'_2 \subset \mathfrak{K}_3 \subset \mathfrak{K}'_3 \subset \mathfrak{K}_1,$$

$$\mathfrak{K}_1 \subset \mathfrak{K}_4 \subset \mathfrak{K}'_4 \subset \mathfrak{K}_1.$$

This gives the desired result. □

As an addendum to Theorem 3.7, Caldwell and Pour-El [PC75] prove that the following definition is also equivalent to those aforementioned:

Definition 3.8. [PR89] (Effective Weierstrass) Let $I^n \subseteq \mathbb{R}^n$ be a closed, bounded rectangle. Namely, we have

$$I^n = \{(x_1, \dots, x_n) : a_i \leq x_i \leq b_i, 1 \leq i \leq n\},$$

where a_i, b_i are computable reals in the sense of definition 3.1. A function

$$f : I^n \to \mathbb{R}$$

is **real computable**[13] if there is a computable sequence of rational polynomials $\langle q_n(x) \rangle$ and a recursive function $g : \mathbb{N} \to \mathbb{N}$ such that for all $x \in I^n$ and all $N \in \mathbb{N}$,

$$m \geq g(N) \implies |f(x) - p_m(x)| \leq 2^{-N}.$$

This definition is particularly useful because it is analytical rather than set theoretic, and thus, makes the task of assessing the computability of specific real functions much more feasible. Indeed, we can show without too much trouble that, in addition to polynomials, many of the continuous transcendental functions that we care about are computable in an arbitrary closed, bounded, computable rectangle [PR89].

Example 3.9. We show that $f(x) = \sin(x)$ is real computable in the sense of definition 3.8. Let $I = [a, b]$, where $a, b \in \mathbb{R}$ with $|a - b| < r$ for some real number r. Let $\langle p_m(x) \rangle_{m \in \mathbb{N}}$ be the well-known rational polynomial approximation to $\sin(x)$. In particular, define

$$p_m(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots + \frac{(-1)^m x^{2m+1}}{(2m+1)!}.$$

Then $\langle p_m(x) \rangle$ is clearly recursive, because we have a well-defined procedure for determining the next polynomial, $p_{m+1}(x)$ from $p_m(x)$. Namely,

$$p_{m+1}(x) = p_m(x) + \frac{(-1)^{m+1} x^{2m+3}}{(2m+3)!}.$$

By Lagrange's remainder theorem [KF75], for all $m \in \mathbb{N}$ there exists some $u \in I$ such that for *all* $x \in I$

$$f(x) = \sin(x) = p_m(x) + \frac{f^{(m+1)}(u)}{(m+1)!} x^{m+1}.$$

[13]Caldwell, Pour-El and Richards use "computable" rather than "real computable" to denote this concept. We will use "real computable" to avoid confusion.

Since $|\sin(x)|, |\cos(x)| \leq 1$ for all x and since x^{m+1} is an increasing function, we have

$$|f(x) - p_m(x)| \leq \frac{b^{m+1}}{(m+1)!}.$$

Stirling's approximation [WW96] gives us

$$n! > \sqrt{2\pi n}\left(\frac{n}{e}\right)^n > \left(\frac{n}{3}\right)^n,$$

so we have

$$\frac{b^{m+1}}{(m+1)!} < \frac{b^{m+1}3^{m+1}}{(m+1)^{m+1}} = \left(\frac{3b}{m+1}\right)^{m+1}.$$

Setting $g(n) = 6bn$, we have, for all $N \in \mathbb{N}$ and all $m \geq g(N)$:

$$|f(x) - p_m(x)| \leq \frac{b^{m+1}}{(m+1)!} \leq \left(\frac{3b}{6bN+1}\right)^{6bN} < \left(\frac{1}{2}\right)^{6bN} < 2^{-N}.$$

So $\sin(x)$ is indeed a real computable function. □

The next obvious question from the analytical perspective is, what happens to computability when we differentiate? Or, as Pour-El and Richards phrase it, "if a computable function possesses a continuous derivative, is the derivative necessarily computable?" [PR89]. Pour-El and Richards demonstrate that the solution to this question depends on whether the second derivative of f, f'', is itself continuous:

Theorem 3.10. [PR89]. *Given a real computable function $f : I \to \mathbb{R}$, where I is as in definition 3.8, if f'' is continuous, then f' is real computable.*

If f'' is *not* continuous then, as Myhill [Myh71] shows, using an example of a function which Pour-El and Richards [PR89] call a "superposition of countably many pulses" of the form:

$$\varphi(x) = \begin{cases} e^{-x^2/(1-x^2)} & : |x| < 1 \\ 0 & : \text{otherwise}, \end{cases}$$

f' is not necessarily real computable.

3.3. **Results.**

Contemporary research in and using recursive analysis has demonstrated its applications to physics, analysis and, to a somewhat lesser degree, computational complexity. See [Wei00] for an in-depth discussion of the recent results of recursive analysis and [BC06] for further contemporary results. Research in recursive analysis has not focused solely on its applications to other fields, however. Bournez and Hainry [BH06] give a description of the techniques of recursive analysis in terms of a class of functions obeying certain criteria. Specifically, Bournez and Hainry demonstrate that real computable functions defined in the sense of Grzegorcyzk and Pour-el and Richards [Grz57, PR89] are precisely those functions contained in the the "smallest class of functions that is...closed by composition, linear integration, minimalization and limit schema" [BH06]. When compared with Grzegorczyk's definition of the class, $\mathfrak{K}$, of real computable functionals (definition 3.5), Bournez and Hainry's conclusion makes intuitive sense. The innovative part of their argument lies in their relation of the technique of linear integration to the definition of what it means to be real computable. The fact that the class of real computable functions is closed under linear integration is particularly interesting in light of the previously cited result of Pour-El, Richards and Myhill (theorem 3.10) that closure does not hold, in general, for the operation of differentiation.

Moore [Moo95] takes the notion of characterizing real computability via integration one step (or perhaps one giant leap) further by *defining* real computability in terms of integrability. Whereas Bournez and Hainry's result, while appealing to the idea of linear integrability, hews to the recursive analysis-supported notion of real computable functions as those which are approximated, in an effective manner, by classically computable functions, Moore's model dispenses altogether with the notion of the necessity of approximabililty. Moore's model defines integration to be a primitive operation of the set of real computable[14] functions. While Bournez and Hainry's result implies that this definition in itself does not separate Moore's model from Recursive Analysis, Moore also introduces an operator, η, which compresses the process of searching over all of $\mathbb{R}$ into a finite-time operation [Moo95]. Moore's model thus creates an interesting link between the approximation focus of Recursive Analysis and the Blum-Smale-Shub model, in which infinite calculations are, by assumption, achievable in finite time.

4. The Blum-Smale-Shub Model

4.1. **Background.** The Blum-Smale-Shub (BSS) model begins from the perspective that, in order to model computation over the real numbers, we need to make the drastic simplifying assumption that computations over infinite-length inputs can be performed in finite time. BSS is not the first model to suggest the introduction of such an axiomatic assumption (BSS does, however, predate the previously mentioned Moore model). In the original description of the BSS model [BSS89], the authors also give a long laundry list of related models of computation, both discrete and continuous. According to them, the models of computation most closely related to the spirit of BSS are [SS63, HI70]. We choose to elaborate on the BSS model in particular because it is both recent (having been described initially in 1989) and influential (having generated a large body of research in the theories of computational complexity and analytical method). We also choose the BSS model because it is described entirely algebraically, and thus serves as an elucidating foil for the analytical, approximation-based description of recursive analysis.

Blum, Smale and Shub see their model as a necessary heir to the "fundamentally" flawed model of classical computation. They (and a fourth author, Cucker) write:

> The point of view of this book is that the [classical] model with its dependence on 0s and 1s is fundamentally inadequate for giving...a foundation to the theory of modern scientific computation, where most of the algorithms–with origins in Newton, Euler, Gauss, et al.–are *real number algorithms* [BCSS97].

Whereas recursive analysis seeks to work within the framework of classical computability to create an analogous notion of computability for real numbers and functions, the BSS model denies the notion that a discrete model should be the foundation of computation in a non-discrete, real-valued world.

4.2. **The model.** We describe the model in the finite dimensional case. That is, we describe a way of computing continuous processes where the inputs and outputs are of the form $\mathbb{R}^n, \mathbb{R}^l$, respectively, for some $n, l \in \mathbb{N}$. It should be noted that, both

[14]Moore uses the alternative terminology of "$\mathbb{R}$-recursive."

in their original exposition of the BSS model and in their subsequent reference, Blum, Smale, Shub and Cucker first define their machines in the more general scenario of computation over an arbitrary ring, **R**. We restrict our definition to the real numbers (which is the ultimate stated goal of the BSS model anyway) for simplicity.

Definition 4.1. A **finite-dimensional BSS machine M over** $\mathbb{R}$[15] is a 10-tuple, $M = (\mathcal{I}_M, \mathcal{S}_M, \mathcal{O}_M, \mathcal{N}_M, t, s_{start}, i, G, H, O)$, which abides by the following criteria:

(1) $\mathcal{I}_M, \mathcal{S}_M$, and $\mathcal{O}_M$ are the **input space**, **output space** and **state space** respectively. For our purposes we restrict $\mathcal{I}_M, \mathcal{S}_M$, and $\mathcal{O}_M$ to be equal to $\mathbb{R}^n, \mathbb{R}^m$ and $\mathbb{R}^l$ respectively.

(2) $\mathcal{N}_M$ is the finite set of nodes through which an input can pass during a M's calculation. Each node in $\mathcal{N}_M$ falls into one of the following four categories, and is associated with a map between or within $\mathcal{I}_M, \mathcal{S}_M, \mathcal{O}_M, \mathcal{N}_M$, as described below:
   (a) The **input node**, s_{start}. s_{start} is the first node through which all inputs pass. s_{start} is associated with a linear map, $i : \mathcal{I}_M \to \mathcal{S}_M$.
   (b) **Computation nodes**. Each computation node, s_n is associated with a rational function $g_{s_n} \in G$, where $g_{s_n} : \mathcal{S}_M \to \mathcal{S}+M$. g_{s_n} is called the **computation map** for s_n. Each computation node s_n has a *unique* outgoing edge, as specified by the transition function, t, below.
   (c) **Branching nodes**. Each branching node, s_n is associated with a polynomial function $h_{s_n} \in H$, where $h_{s_n} : \mathcal{S}_M \to \mathbb{R}$. h_{s_n} is called the **branching map** for s_n. Branching nodes are effectively a method of making comparisons among elements of the state space $\mathcal{S}$, as specified by the transition function, t, below.
   (d) **Output nodes**. Each output node, s_n is associated with a linear map $o_{s_n} \in O$, where $o_{s_n} : \mathcal{S} \to \mathcal{O}$. o_{s_n} is, unsurprisingly, called the **output map** for s_n. Output nodes have no outgoing edges to other nodes within the graph.

(3) The **transition function** for M, $t : \mathcal{N}_M \times \mathcal{S}_M \to \mathcal{N}_M$, gives the next node through which a given input will pass. For a node s_n that is either the start node or a calculation node, $t(s_n, x) = s_{n'}$ for all $x \in \mathcal{S}_M = \mathbb{R}^m$. So s_n has a single, determined next node, $s_{n'}$. For s_n a branch node, we have:

$$t(s_n, r) = \begin{cases} s_{n^+} & : h_{s_n}(x) \geq 0 \\ s_{n^-} & : h_{s_n}(x) < 0, \end{cases}$$

Where s_{n^+} and s_{n^-} are possibly (and, in practice, always) non-equivalent nodes, and h_{s_n} is the branching function associated to h_{s_n}. If s_n is an output node, then s_n has no outgoing edges to other nodes. For completeness of terminology, we thus define $t(s_n, x) = s_n$ in this case.

We observe several features of this definition. Firstly, note that, without loss of generality, we may assume that M has a unique output node n_{out} (we could just add an extra node at the end of the computation into with all the output nodes mapped that does nothing but output whatever it is given). Also note that a BSS machine is, like a Turing machine, effectively a finite graph, where the directed

[15][BSS89] and [BCSS97] refer only to finite dimensional *machines* over $\mathbb{R}$. We insert "BSS" for clarity.

edges between the nodes are given by a transition function. However, the criteria for an allowed edge within a BSS machine is considerably more restrictive than in the Turing machine case. Whereas a given node in a Turing machine may have outgoing edges to at most $|\Gamma|$ nodes, where Γ is the tape alphabet (see definition 2.2), a node within a BSS machine has at most 2 outgoing edges. Except for in the case of branching nodes, the present node *uniquely* determines the next position of the machine.

Relatedly, whereas in the definition of the Turing machine (definition 2.2), the state space, S, defined the set of nodes for the graph of T, BSS machines have both states *and* nodes. For the BBS machine, while there are only a finite number of nodes, there are an uncountable number (all $x \in \mathbb{R}^m$) of states in which a calculation could reside at any given step. Herein lies the fundamental difference between Turing machines and BSS devices. Whereas in a Turing machine, the calculation takes place in the transitions between edges, in a BSS machine, the calculations take place *within* the nodes. The edges between the nodes of a BSS machine are essentially a way to separate different, but inevitably sequential, steps of calculation.

Also note that, while M essentially recreates the action of a function or process from $\mathbb{R}^n \to \mathbb{R}^l$, the transition functions describing the interim steps *must* be of degree m, where m may or may not be equal to either n or l. By assuring that the input map i and the output maps O are linear, we assure that all of the (meaningful) calculations and comparisons for M occur among ration functions on $\mathbb{R}^m$.

Example 4.2. We present an example of the types of questions that can be answered with the BSS model. We describe (and depict, in figure 2) a BSS machine that simulates Steffensen's method,[16] an algorithm that approximates roots of polynomial functions over $\mathbb{R}$. Steffenson's method works by making successively better guesses for the value of the root based on linear approximations. Suppose $f : \mathbb{R} \to \mathbb{R}$ is a polynomial with a has a root that we wish to approximate. Choose an arbitrary $x_0 \in \mathbb{R}$ to be our first guess. Then x_{n+1} is defined as follows:

$$x_{n+1} = x_n - f(x_n) \cdot \frac{f(x_n)}{f(x_n + f(x_n)) - f(x_n)} = x_n - \frac{(f(x_n))^2}{f(x_n + f(x_n)) - f(x_n)}.$$

Instead of using f', as in Newton's method, to find the linear approximation, we *approximate* f' via the formula,

$$\frac{f(x_n)}{f(x_n + f(x_n)) - f(x_n)},$$

If, as we hope, $f(x_n) \to 0$ as $n \to \infty$, this formula gives an increasingly accurate approximation of the derivative $f'(x_n)$. The sequence $\langle x_n \rangle$ described by Steffensen's method converges to the root under the mildly restrictive conditions[17] that f has a continuous second derivative f'' in a closed interval of length 2α centered around the root, and that $f'(\alpha) \neq 0$.

We chose to describe the BSS machine for Steffensen's method because it emphasizes the discount that the BSS applies to classically time-intensive calculations.

[16]Blum, Smale, Shub and Cucker [BCSS97] present a similar example of a BSS machine which simulates Newton's method.

[17]See [DB03, BF04] for a more in-depth description of Steffensen's method and its criteria for convergence.

Namely, in order to calculate x_{n+1} from x_n, we must perform at least two calculations on f: first, we must calculate $f(x_n)$, and then, using this result, we must calculate $f(x_n + f(x_n))$. For a high-degree polynomial f and an irrational number x_n, this is a complicated task; however, this potential snare is not of concern within the BSS model. Such polynomial computations are, by assumption, accomplished in constant time. Instead of bothering with questions about how long it takes to do a given calculation, the BSS model is more concerned with the speed of a given algorithm's convergence in a global sense. That is, how many *steps* of calculation do we have to do to get an output? We know from [DB03] that Steffensen's method (like Newton's method [BCSS97]) converges quadratically.

The approximation of a root to an arbitrary precision for f via Steffensens's method is easily represented with the following BSS Machine, N. In this case, $\mathcal{I} = \mathcal{S} = \mathcal{O} = \mathbb{R}$. Let $\varepsilon > 0$ and let N be give by figure 2[18],

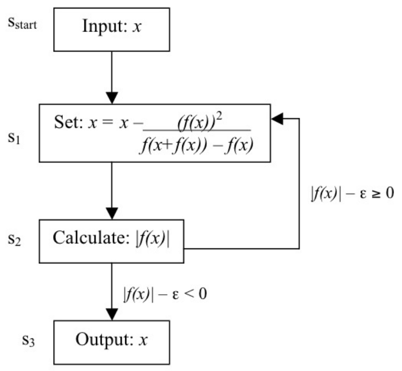

FIGURE 2. A Steffensen's method machine to approximate a root of f to within ε.

The transition function is clearly specified by the edges between the nodes given above. Note that the "=" in node s_1 above is an *assignment* rather than an equivalence. s_1 is a computation node which effectively resets the value of the the current input x based on the next value of the sequence $\langle x_n \rangle$ given above. This gives

$$g_{s_1}(x) = x - \frac{(f(x))^2}{f(x + f(x)) - f(x)}.$$

The calculation branches in s_2 based on the real-valued branching function h_{s_2} given by

$$h_{s_2}(x) = |f(x)| - \varepsilon.$$

If $h_{s_2} < 0$, meaning $f(x)$ is less than ε away from being a root, then we output the x that we have found. Otherwise, we do another round of Steffensen's algorithm and try again. It is worth noting that, just as when we perform Steffensen's method by hand, the recursive nature of the N lends itself to the possibility that, on a given input x, we will never output anything, and will instead be stuck in an infinite loop of calculation. Blum, Smale, Shub and Cucker define Ω_T ($T \in \mathbb{N}$), which they call

[18]I drew figure 2 myself, based on a similar figure for the Newton machine in [BCSS97].

the **time-T halting set** of N, to be the set of *all* x which have output a result after at most T trips through the branching node s_2. More generally, they define the **halting set** of N, Ω by

$$\Omega = \bigcup_{T \in \mathbb{N}} \Omega_T.$$

Ω is the set of all inputs x on which N eventually reaches an output.

The notion of the halting set for N leads to the BSS conception of computability[19], which, unsurprisingly, is exactly analogous to the classical definition:

Definition 4.3. [BCSS97] A set[20] S is **BSS computable** if there exists a BSS machine, M_S that describes it, such that the halting set for M is equal to the output space. That is,

$$\Omega_S = \mathcal{O}_{M_S}.$$

A set is **semicomputable** if there exists a BSS machine M_S such that:

$$M_S(x) = \begin{cases} 1 & : x \in S \\ 0 \text{ or undefined} & : \text{otherwise.} \end{cases}$$

It is a simple fact that, if a set S and its complement S^c are both semicomputable, then S is BSS computable. The proof is analogous to the classical case (see [Sip05] for a proof in the classical case).

As previously noted, Steffensen's approximation is clearly not BSS computable for all possible polynomial functions f. Take $f(x) = 1 - x^2$. Then, f has roots at $x = \pm 1$, Letting N_f be defined as above, $N_f(0)$ never produces an output since it is stuck in an infinite loop of $x = 0$. Hence $0 \notin \Omega$. Since nothing precludes our choosing 0 as an input, N_f cannot be BSS computable. Nonetheless, there are other interesting questions that we can ask about Steffensen's approximation with respect to BSS computability. Specifically, we know that Steffensen's approximation converges for some inputs, but not for others, so we might wonder, as [BCSS97] notes for Newton's method, whether the set of input for which Steffensen's approximation converges for a given function f is *itself* computable. The answer to this question is analogous to the case for Newton's method, which is discussed in [BCSS97]. In the case of Newton's method, the answer depends on the computability of the Julia set, which is beyond the scope of this exposition.

4.3. **Results.** Blum, Smale, Shub and Cucker give a comprehensive review of the research on and using BSS models through 1997 in [BCSS97]. One example that they cite in which techniques of BSS model have been useful, that is, the discussion of the computability and complexity of the Mandelbrodt set. For each $c \in \mathbb{C}$[21], we define the function $f_c : \mathbb{C} \to \mathbb{C}$ by

$$f_c(z) = z^2 + c.$$

[19][BCSS97] uses the analogous word, "decidability."

[20][BCSS97] uses the word "problem" to describe more general scenarios like Steffensen's method of approximating roots. When asking questions about BSS computability for such an object, we are referring to the computability of the sets that the object defines. The BSS model does not have a notion of computability for functions that is analogous to real computability for functions as described in section 3.2

[21]To this point, we have spoken about computability within both the recursive analysis and BSS framework only for $\mathbb{R}^n$. Considering $\mathbb{C}$ as $\mathbb{R}^2$, the definition of computability over $\mathbb{C}$ is clear.

and define the sequence $\langle c_n \rangle_{n\in\mathbb{N}}$ to be the sequence given by $c_0 = c$, $c_{n+1} = f_c(c_n)$. Then the Mandelbrodt set [BCSS97, BC06] is the set of points:

$$\mathcal{M} = \{c \in \mathbb{C} : \langle c_n \rangle \text{ converges.}\}$$

Blum, Smale and Shub [BSS89] seek to answer the question of whether the Mandelbrodt set is computable within their framework. That is, can we find a BSS machine M such that $M(x) = 1$ for all $x \in \mathcal{M}$ and $M(x) = 0$ otherwise. They point out that the BSS model is particularly suited to this question because of its infinite precision. In contrast to the recursive analysis model, where, as we demonstrated in fact 3.3, it is sometimes impossible to make definitive, effective comparisons between quantities, within the BSS model, we do not have such a problem [BCSS97]. See section 5 for more on this particular difference between the BSS model and recursive analysis.

Blum, Smale and Shub demonstrate that the Mandelbrodt set is *not* computable by showing that its complement,

$$\mathcal{M}^c = \{c \in \mathbb{C} : c_n \to \infty \text{ as } n \to \infty\}.$$

is not computable (if either $\mathcal{M}$ or $\mathcal{M}^c$ were computable by a machine M, then the other set would automatically be computable by M' defined to be M with the output states flipped).

The proof of the uncomputability of M^c relies on a class of objects fundamental to the BSS model: semi-algebraic sets.

Definition 4.4. Let $\{R_1, R_2, \ldots, R_n\}$ be a finite system of polynomial equations and/or inequalities over $\mathbb{R}$ (so $R_i(x) = 1$ if x satisfies R_i and 0 otherwise). A set $S \in \mathbb{R}^n$ is **basic semi-algebraic over** $\mathbb{R}$ if

$$S = \{x \in \mathbb{R}^n : \forall i < n[R_i(x) = 1]\}.$$

A set S is **semi-algebraic** if S is the finite union of basic semi-algebraic sets.

The proof, as given by [BCSS97] that the $\mathcal{M}$ is not computable is then sketched as follows:

Lemma 4.5. [Shi98] $\mathcal{M}$ *is not the countable union of semi-algebraic sets over* $\mathbb{R}$

Blum, Smale, Shub and Cucker cite this lemma without proof, claiming that it lies within the purview of complex dynamics and is outside of the scope of real computation. They pause to note that the proof is a consequence of $\mathcal{M}$ having Hausdorff dimension 2. Informally, this means that we can cover $\mathcal{M}$ with a collection of balls $\langle B_i(r_i) \rangle_{i\in I}$ for some I such that the sum of all the r_i is less than 2, but that we cannot find such a covering for any number less than 2. The proof that $\mathcal{M}$ is not computable is then completed by the following theorem:

Theorem 4.6. [BCSS97] *If M is a BSS machine over $\mathbb{R}$, then*

1. *For any $T > 0$, Ω_T is a finite disjoint union of basic semi-algebraic sets,*
2. *The halting set Ω_M is a countable disjoint union of basic semi-algebraic sets.*

The proof of this theorem depends on defining Ω_T in a new way. Let M be a BSS machine, and γ be a logical path through the machine of possibly infinite length. Define $\gamma(k)$ to be the first k steps of γ, and let $\nu_{\gamma(k)}$ be the set of points in $\mathcal{I}_M$ that have computations paths that share the first k steps with γ. Symbolically,

$$\nu_{\gamma(k)} = \{x \in \mathcal{I}_M : \gamma_x(k) = \gamma(k)\}.$$

Relatedly, define Γ_T to be the set of all paths through M that halt in time T, and let Γ_M be the union of all such Γ_T. Define the slightly smaller set, Γ'_M by to be the set of paths through that reach the output node, n_{out} exactly once. Then Blum, Smale, Shub, and Cucker observe that

$$\Omega_T = \bigcup_{\gamma \in \Gamma_T} \nu_\gamma, \text{ and } \Omega_M = \bigcup_{\gamma \in \Gamma'_M} \nu_\gamma.$$

The desired result follows by observing the "natural correspondence" between semi-algebraic sets and semi-algebraic formulas [BCSS97]. Hence, $\mathcal{M}$ is not described by any BSS machine, and is thus not computable. Aside from addressing questions of set computability, the BSS model has also been influential in complexity theory because of it's relation to algebraic circuits (see [BCSS97, AB09]) for a description of the BSS model's uses in computational complexity theory.

5. Comparisons

So, which model is the best? Before addressing this question directly, we observe several important distinctions that have emerged during the past 18 pages. The first relates to the question of what is "easy" to compute with respect to each model. Recall example 3.9, in which we demonstrated that the function

$$f(x) = \sin(x)$$

is real computable. Similar proofs can be given for a variety of transcendental functions, including all the trigonometric functions, e^x, and $\Gamma(x)$ [PR89]. These proofs all rely on the fact that recursive analysis is fundamentally a model of real computation based on approximation by rations, and the functions in question are all approximated by well-known sequences of polynomials. Try proving the same result for BSS computability and you run into a snag. While the BSS model has, in some sense, greater computational power because it assumes the compression of infinite computations into finite space, it only defines such power with respect to *polynomial* functions. The BSS model's great strength, that it algebratizes the theory of computation to the greatest degree possible and, in the process, dovetails with contemporary research directions in computational complexity, is also its weakness when it comes to describing computations on other classes of functions.[22] Moore [Moo95] points out this flaw in the BSS model, and uses it as evidence for the validity of his own model of analog computability, which is based on the operation of function integration, and thus includes the aforementioned transcendental functions as primitives.

Before getting too down on the BSS model for its apparent inability to model some of the functions we care about most, we note an important realm in which the BSS model easily surpasses recursive analysis: precision in comparison. Recall fact 3.3, which states that, within the framework of recursive analysis, it is possible to define a number $x = 0$ and have no effective procedure for demonstrating that $x = 0$. This implies the existence of scenarios in recursive analysis in which we would be unable to effectively compare two quantities. Using the BSS model, we have no such problem. See figure 3[23].

[22][BC06], also points out this distinction between the BSS and the recursive analysis model (which he calls the "bit-model").

[23]I drew this figure myself based on a similar figure in [BCSS97].

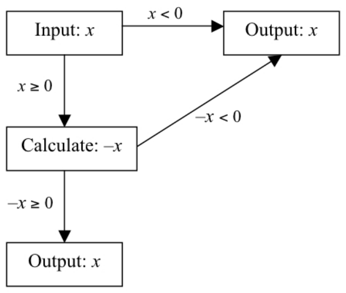

Figure 3. A BSS subroutine for determining whether the quantity x is equal to 0. Because the BSS model works under the assumption of infinite precision, it is able to determine, *definitively* and in finite time, whether two quantities are equal. This is a feat which the approximation-based recursive analysis model could never hope to achieve.

6. Conclusion

The BSS model and recursive analysis are designed for different, albeit overlapping purposes. The BSS model, with its foundation in algebraic equations and assumption of infinite precision, is invaluable for giving evidence on questions of set computability and computational complexity. In contrast, recursive analysis is better at dealing with questions about functional computability. Additionally, because it focuses on rational approximation, recursive analysis is perhaps a more realistic model of the way that we think about, or at least program our computers to think about, computing on irrational numbers.

This dichotomy, and the volume of research generated for each model independently speaks to the lack of consensus over exactly what is meant by the phrase "continuous computation." Do we believe that this means infinite precision or decimal approximation? The answer depends on what you are trying to prove.

7. Acknowledgments

I would like to express my profound gratitude to my advisor, Dr. Gerald Sacks, for his mentorship, both in learning mathematical logic and in finding the best path from Minsk to Pinsk. I would also like to thank Hannah Resnick and David Kosslyn for their critique and proofreading of earlier drafts. Finally, I would like to thank my parents, Raiselle and Kenneth Resnick, for twenty-two years of support and guidance and for believing that I could do hard math. ♡

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 1 edition, April 2009.
[Bar05] John D. Barrow. How to do an infinite number of things before breakfast. *New Scientist*, (2484):28–32, January 2005.
[BC06] Mark Braverman and Stephen Cook. Computing over the reals: Foundations for scientific computing. *Notices of the AMS*, 53, 2006.

[BCSS97] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation.* Springer, 1 edition, October 1997.

[BF04] Richard L.(Richard L. Burden) Burden and J. Douglas Faires. *Numerical Analysis.* Brooks Cole, 008 edition, December 2004.

[BH06] Olivier Bournez and Emmanuel Hainry. Recursive analysis characterized as a class of real recursive functions. *Fundamenta Informaticae*, 74(4):409–433, January 2006.

[Bra97] Vasco Brattka. Order-free recursion on the real numbers. *Mathematical Logic Quarterly*, 43(2):216–234, 1997.

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

[Dav65] Martin Davis. *The undecidable; basic papers on undecidable propositions, unsolvable problems and computable functions.* Raven Press, Hewlett, N.Y.,, 1965.

[DB03] Germund Dahlquist and Ake Bjorck. *Numerical Methods.* Dover Publications, April 2003.

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

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

[HI70] Gabor T. Herman and Stephen D. Isard. Computability over arbitrary fields. *J. London Math. Soc.*, 2(2):71–79, 1970.

[KF75] A. N. Kolmogorov and S. V. Fomin. *Introductory Real Analysis.* Dover Publications, 1st edition, June 1975.

[Kle52] Stephen Cole Kleene. *Introduction to metamathematics.* University series in higher mathematics. Van Nostrand, New York,, 1952.

[Koi97] Pascal Koiran. A weak version of the blum, shub, and smale model. *Journal of Computer and System Sciences*, 54:177189, February 1997. ACM ID: 255841.

[LS01] Roger C. Lyndon and Paul E. Schupp. *Combinatorial Group Theory.* Springer, March 2001.

[Moo95] Cristopher Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science*, 162:23—44, 1995.

[Moo96] Cristopher Moore. Dynamical recognizers: Real-time language recognition by analog computers. *Theoretical Computer Science*, 201:99—136, 1996.

[Mos09] Yiannis N. Moschovakis. *Descriptive Set Theory.* American Mathematical Society, 2 edition, June 2009.

[Myh71] J Myhill. A recursive function, defined on a compact interval and having a continuous derivative that is not recursive. *Michigan Math. J*, 18(2):97–98, 1971.

[PC75] Marian Boykan Pour-El and Jerome Caldwell. On a simple definition of computable function of a real variable-with applications to functions of a complex variable. *Mathematical Logic Quarterly*, 21(1):1–19, 1975.

[PR89] Marian Boykan Pour-El and J. Ian Richards. *Computability in Analysis and Physics.* Springer-Verlag, 1989.

[Shi98] Mitsuhiro Shishikura. The hausdorff dimension of the boundary of the mandelbrot set and julia sets. *The Annals of Mathematics*, 147(2):225–267, March 1998. ArticleType: research-article / Full publication date: Mar., 1998 / Copyright 1998 Annals of Mathematics.

[Sip05] Michael Sipser. *Introduction to the Theory of Computation.* Course Technology, 2 edition, February 2005.

[SS63] J. C Shepherdson and H. E Sturgis. Computability of recursive functions. *Journal of the ACM (JACM)*, 10:217255, April 1963. ACM ID: 321170.

[Tur36] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2):230–65, July 1936.

[Wei95] Klaus Weihrauch. A simple introduction to computable analysis. *Informatik Berichte Nr.*, 171:1–80, 1995.

[Wei97] Klaus Weihrauch. A foundation for computable analysis. In *Proceedings of the 24th Seminar on Current Trends in Theory and Practice of Informatics: Theory and Practice of Informatics*, SOFSEM '97, pages 104–121. Springer-Verlag, 1997.

[Wei00] Klaus Weihrauch. *Computable analysis: an introduction.* Springer, November 2000.

[WW96] E. T. Whittaker and G. N. Watson. *A Course of Modern Analysis.* Cambridge University Press, 4 edition, September 1996.

[Zho98] Ning Zhong. Recursively enumerable subsets of rq in two computing models Blum-Shub-Smale machine and turing machine. *Theoretical Computer Science*, 197(1-2):79–94, May 1998.