
A RECURSION THEORETIC FOUNDATION OF COMPUTATION OVER REAL NUMBERS

KENG MENG NG, NAZANIN R. TAVANA AND YUE YANG

Division of Mathematical Sciences, School of Physical & Mathematical Sciences, Nanyang Technological University, 21 Nanyang Link, Singapore 637371

Amirkabir University of Technology, 424 Hafez Ave, Tehran, Iran, P.O. Box: 15875-4413

Department of Mathematics, National University of Singapore, Block S17, 10 Lower Kent Ridge Road, Singapore 119076

e-mail address: kmng@ntu.edu.sg

e-mail address: nrtavana@aut.ac.ir

e-mail address: matyangy@nus.edu.sg

ABSTRACT. We define a class of computable functions over real numbers using functional schemes similar to the class of primitive and partial recursive functions defined by Gödel [3, 4] and Kleene [9]. We show that this class of functions can also be characterized by master-slave machines, which are Turing machine like devices. The proof of the characterization gives a normal form theorem in the style of Kleene [9]. Furthermore, this characterization is a natural combination of two most influential theories of computation over real numbers, namely, the type-two theory of effectivity (TTE) (see, for example, Weihrauch [18]) and the Blum-Shub-Smale [1] model of computation (BSS). Under this notion of computability, the recursive (or computable) subsets of real numbers are exactly effective Δ_2^0 sets.

1. INTRODUCTION

The original motivation of this paper was to understand the notion of algorithm in its general form, i.e., not necessarily over domain \mathbb{N} — the set of natural numbers¹. By studying algorithms over real numbers and making comparisons with their counterpart over \mathbb{N} , we came across a notion of computability over real numbers—actually two similar notions, one over Baire Space² and the other over \mathbb{R} identified as the set of equivalence classes of Cauchy sequences. The most interesting feature of this notion is its resemblance to the classical notion of “computability over natural numbers”. In fact, it can be viewed as a natural lift of the classical notion from many different aspects (recursive definitions, machine model

The first author is partially supported by the MOE grant MOE-RG26/13. The last author is partially supported by MOE grant MOE-2019-t2-2-121.

¹In this paper, we use both \mathbb{N} and ω to denote the set of natural numbers.

²In this paper, we use both \mathcal{N} and ω^ω to denote the Baire space.

and definability). Before we elaborate the details, let us make some short comments on the study of algorithms and how it motivated us.

The concept of algorithm was formalized in 1930s. It is one of the greatest intellectual achievements in history. In fact there are several equivalent formalizations based on different insights on effectiveness. They all apply to the domain of natural numbers or objects which can be coded by natural numbers. Furthermore, the formalizations themselves can be formulated within first-order Peano arithmetic. All formulations captured the finite and discrete nature of algorithms, and revealed the intrinsic link between computability and natural numbers.

However, the informal concept of algorithms applies not only the natural numbers but also to other domains. For example, Newton's method of finding roots is an algorithm over real numbers and real-valued functions. Algorithms are used in other areas of natural sciences too, for example, laboratory procedures for experimental scientists. Most people would agree that (at least for a fixed domain) there is a clear intuition about what algorithms are. What we hope to formalize is "that clear intuition" over real numbers.

Among the informal descriptions of algorithm, the one below gives the best illustration of our approach:

Definition 1.1 (Informal). An *algorithm* is a finite set of instructions such that

- (1) each instruction is "effective" (that is, definite or clearly stated or sufficiently simple etc);
- (2) there is a clear organization of the instructions so that from input to output, we know how to go from one instruction to another.

What we like about this description is the separation of the instruction part and the overall organization part. The instructions in item (1) may depend on domains and one needs to justify the effectiveness of them; whereas the organization part in item (2) is really the heart of algorithms, and it should be "absolute" when we move from one domain to another.

When the domain is the set of real numbers, part (1) actually depend on the underlying topology, that is why we consider both the Baire space ω^ω and the real numbers \mathbb{R} . However, part (2) is done in the same way as over \mathbb{N} . Over each of the domains, we formalize the notion of computability in two ways, namely, by functional schemes and by machines; and show that the two formalizations actually give us the same class of functions. We will come back to the discussion of algorithms in the concluding section.

2. COMPUTABILITY THEORY OVER BAIRE SPACES

We choose Baire space $\mathcal{N} = \omega^\omega$ as our first example, because it will shed light on computation over real numbers while avoiding the tacky issues like dealing with equivalence classes of Cauchy sequences. It is the first step of formalizing computability on higher types, an area closer to logicians than computational mathematicians.

As we observed earlier, computability and natural numbers are intrinsically linked together. Therefore, we always need to have natural numbers at our disposal. For this reason, when we talk about computability over a fixed domain D , we consider the class of functions \mathcal{F} from mixed types to mixed types, more precisely, \mathcal{F} consists of functions from $Y \rightarrow Z$ where $Y \in \{\mathbb{N}^p \times D^q, \mathbb{N}^p, D^q\}$ and $Z \in \{\mathbb{N}, D\}$, where p and q are positive integers. In particular functions from \mathbb{N} to \mathbb{N} and from D to D are included. As functions with codomain

$\mathbb{N}^p \times D^q$ can be viewed as juxtaposition of $p + q$ many functions in \mathcal{F} , we will not study them explicitly.

For notational simplicity, we sometimes discuss only functions with domain $\mathbb{N} \times D$ instead of $\mathbb{N}^p \times D^q$. Note that for domains like Baire space, \mathbb{N} can be naturally embedded into them. However, working with mixed types is necessary in more general domains, for example, computation on finite rings. In the remaining of this section, the domain D will be the Baire space \mathcal{N} , and $\mathcal{F} = \{f : f \text{ is a function from } \mathbb{N}^p \times \mathcal{N}^q \text{ to either } \mathbb{N} \text{ or } \mathcal{N}\}$.

2.1. Formalizing computability using functional schemes. We begin with a concept used by computable analysts, which can be traced back to the notion of relativised computation by Turing [16].

Definition 2.1. We say that a partial function $F : \mathcal{N} \rightarrow \mathcal{N}$ is *TTE-computable* over \mathcal{N} if there is an oracle Turing machine M such that for all x in \mathcal{N} , $F(x) \downarrow = y$ if and only if for all i in \mathbb{N} , $M^x(i) \downarrow = y(i)$; and $F(x) \uparrow$ if and only if for some i in \mathbb{N} , $M^x(i) \uparrow$. (Here we used the standard notation that M^x is the machine M with x as its oracle, and \downarrow means “is defined” and \uparrow means “is undefined”.)

One can also define TTE-computable functions as induced by recursive functions $f : \omega^{<\omega} \rightarrow \omega^{<\omega}$ satisfying the usual monotonicity conditions. We use oracle Turing machines because of their direct connection with master-slave machines which we define later. It is well-known that universal oracle Turing machine exists (see for example, Soare [14] p.48). Thus we have

Lemma 2.2. *There is a universal TTE-computable function $\Psi(e; x)$ over \mathcal{N} , i.e., for any TTE-computable function $F : \mathcal{N} \rightarrow \mathcal{N}$, there is some $e \in \mathbb{N}$ such that for all $x \in \mathcal{N}$, $F(x) = \Psi(e; x)$.*

Here is our first main definition:

Definition 2.3. The class of *partial recursive functions over \mathcal{N}* is the smallest subclass \mathcal{C} of \mathcal{F} satisfying the following conditions:

- (1) \mathcal{C} contains the following basic functions:
 - (a) Zero function $Z : \mathbb{N} \rightarrow \mathbb{N}$, $Z(n) = 0_{\mathbb{N}}$;
 - (b) successor function $S : \mathbb{N} \rightarrow \mathbb{N}$, $S(n) = n + 1$; and
 - (c) for natural numbers p, q and i with $p + q \geq 1$ and $1 \leq i \leq p + q$ the projection function
$$\pi_i^{p+q}(n_1, \dots, n_p; x_1, \dots, x_q) = \begin{cases} n_i, & \text{if } i \leq p; \\ x_{i-p}, & \text{if } i > p. \end{cases}$$
 - (d) A universal TTE-computable function $\Psi(e; x)$ over \mathcal{N} ; and
 - (e) the characteristic function $\chi : \mathcal{N} \rightarrow \mathbb{N}$ of $\{0_{\mathcal{N}}\}$ where $0_{\mathcal{N}}$ is the constant zero sequence in Baire space.
- (2) \mathcal{C} is closed under
 - (a) composition, provided the types are matched;
 - (b) primitive recursion with respect to \mathbb{N} ; and
 - (c) μ -operator with respect to \mathbb{N} .

The class of *primitive recursive functions over \mathcal{N}* is defined similarly with the following changes: (i) Fix a recursive list of indices $e_i, i = 0, 1, \dots$ such that the Turing machine M_{e_i}

computes the i -th classical primitive recursive function. Define the universal primitive TTE function $\Theta(i; x)$ as $\Theta(i; x) := \Psi(e_i; x)$. Replace Ψ in item (1)(d) by Θ ; (ii) Drop the clause (2)(c). We say a predicate is primitive recursive over \mathcal{N} if its characteristic function is a primitive recursion function over \mathcal{N} . Observe that all primitive recursive over \mathcal{N} functions are total.

We now give precise definition of the terminologies used in the closure properties (item (2) in Definition 2.3). In the definitions below, $\vec{n}, \vec{m}, \vec{x}$ and \vec{y} stand for the tuples (n_1, \dots, n_p) , (m_1, \dots, m_r) , (x_1, \dots, x_q) and (y_1, \dots, y_s) respectively.

Definition 2.4. Given a function $f(n_1, \dots, n_p; x_1, \dots, x_q)$ and a finite sequence of functions $g_i(\vec{m}; \vec{y})$ and $h_j(\vec{m}; \vec{y})$ where $1 \leq i \leq p$ and $1 \leq j \leq q$, we say that the types of f , g_i and h_j are *matched* if the codomain of g_i is \mathbb{N} and the codomain of h_j is \mathcal{N} .

We say that a class of functions \mathcal{C} is *closed under composition, provided the types are matched*, if for functions $f(\vec{n}; \vec{x})$, $g_i(\vec{m}; \vec{y})$ and $h_j(\vec{m}; \vec{y})$ with matched types ($1 \leq i \leq p$ and $1 \leq j \leq q$), f , g_i and h_j are in \mathcal{C} implies the function

$$f(g_1(\vec{m}; \vec{y}), \dots, g_p(\vec{m}; \vec{y}); h_1(\vec{m}; \vec{y}), \dots, h_q(\vec{m}; \vec{y}))$$

is in \mathcal{C} .

Definition 2.5. Let $G : \mathbb{N}^p \times \mathcal{N}^q \rightarrow \mathcal{N}$ and $H : \mathbb{N}^{p+2} \times \mathcal{N}^q \times \mathcal{N} \rightarrow \mathcal{N}$ be given. We say that a function $F : \mathbb{N}^{p+1} \times \mathcal{N}^q \rightarrow \mathcal{N}$ is obtained from G and H *by primitive recursion* with respect to \mathbb{N} , if

$$\begin{aligned} F(0, \vec{n}; \vec{x}) &= G(\vec{n}; \vec{x}) \\ F(k+1, \vec{n}; \vec{x}) &= H(k, \vec{n}; \vec{x}, F(k, \vec{n}; \vec{x})). \end{aligned}$$

Similarly for function $F : \mathbb{N}^{p+1} \times \mathcal{N}^q \rightarrow \mathbb{N}$ (just change the codomains of G and H to \mathbb{N} and move the term $F(k, \vec{n}; \vec{x})$ before the semicolon in the second equation).

Definition 2.6. Let $G : \mathbb{N}^{p+1} \times \mathcal{N}^q \rightarrow \mathbb{N}$ be given. We say that a function $F : \mathbb{N}^p \times \mathcal{N}^q \rightarrow \mathbb{N}$ is obtained from G *by μ -operator* with respect to \mathbb{N} , if

$$F(\vec{n}; \vec{x}) = \begin{cases} \text{the least } k \text{ such that for any } k' < k, & \text{if such } k \text{ exists;} \\ G(k', \vec{n}, \vec{x}) \downarrow \neq 0_{\mathbb{N}} \text{ and } G(k, \vec{n}, \vec{x}) = 0_{\mathbb{N}}, & \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

We write $F(\vec{n}; \vec{x})$ as $\mu k G(k, \vec{n}; \vec{x}) = 0$.

Next we state a few lemmas which will be needed later. Only sketches of the proofs are given, as they are almost entirely the same as the standard proofs. Note that every classical recursive function on \mathbb{N} has a natural pointwise lift on \mathcal{N} . For example, the lift of exponential function $n \mapsto 2^n$ is: $(n_0, n_1, \dots) \mapsto (2^{n_0}, 2^{n_1}, \dots)$. We understand that this lift is not the usual exponential function on real numbers, that is why we need to go beyond Baire space later.

Lemma 2.7. *All partial recursive functions over natural numbers in the standard sense are partial recursive over \mathcal{N} . In particular, Gödel numbering of finite sequences of natural numbers and the corresponding decoding functions are primitive recursive over \mathcal{N} .*

Proof. By definition, it is clear that the generalized recursive function over \mathcal{N} contains the standard partial recursive functions as a subclass. \square

Lemma 2.8. *The equality predicate “ $x = y$ ” over \mathcal{N} is primitive recursive over \mathcal{N} .*

Proof. By Lemma 2.7, the cutoff subtraction $x \dot{-} y$ and the absolute value function $|x - y| = (x \dot{-} y) + (y \dot{-} x)$ are TTE-computable functions over \mathcal{N} . Therefore $x = y$ if and only if $\chi(|x - y|) = 1$. The result follows. \square

Lemma 2.8 explains the purpose of having the characteristic function χ of $\{0_{\mathcal{N}}\}$: to make equality computable.

Lemma 2.9. *If $f : \mathbb{N} \times \mathcal{N} \rightarrow \mathbb{N}$ is primitive recursive over \mathcal{N} , then so are $\sum_{i=0}^n f(i, x)$ and $\prod_{i=0}^n f(i, x)$. Consequently the class of primitive recursive over \mathcal{N} predicates is closed under bounded (numerical) quantifiers.*

Proof. The proof is exactly the same as the one in classical recursion theory. \square

Lemma 2.7 and 2.9 also explains why we must include \mathbb{N} in the formalizations of computability, because we need the coding of finite sequences³.

Lemma 2.10 (definition by cases). *If $f_1(x)$ and $f_2(x)$ are primitive recursive functions over \mathcal{N} , and $P(x)$ and $Q(x)$ are mutual exclusive primitive recursive predicates over \mathcal{N} , then the function*

$$f(x) = \begin{cases} f_1(x), & \text{if } P(x) \text{ holds;} \\ f_2(x), & \text{if } Q(x) \text{ holds.} \end{cases}$$

is also primitive recursive over \mathcal{N} .

2.2. Formalizing Computability by Master-Slave Machines. We now adopt the master-slave machines (abbreviated as MS-machines) used in Yang [19] to formalize the computability over \mathcal{N} . For the sake of simplicity, we will fully employ Church’s Thesis of computation over natural numbers.

Let us quickly recall the basic features of a master-slave machine (see Figure 2.2). Its physical device consists of the following three parts: Master part (above the dashed line in Figure 2.2) which only handles natural numbers; slave part (below the dashed line) which handles computations over Baire space; and their interactions (on the dashed line) where the information flows from the master to slaves and vice versa.

The master part is simply a standard Turing machine M . As it is the control part of the machine, the same letter M is also used for the whole MS-machine. If a computation involves natural number inputs, they will be sent to the master M as well.

Although one can view the slave part as a big black box, the description below gives more intuition to the computation, and its effectiveness. We have an infinite sequence $\langle S_i : i \in \omega \rangle$ of universal (standard) oracle Turing machines, called *slaves*. Each slave S_i has its own label i , indicating its relative position to other slaves. For an element $x = (m_0, m_1, m_2, \dots)$ in Baire space, we put x on tape as $b(m_0) \square b(m_1) \square b(m_2) \square \dots$, where $b(m)$ stands for m written in binary and \square stands for empty cell. All slaves share the following four kinds of tapes: (1) An oracle tape on which the input x is written, we sometimes refer to it as the *input tape* for slaves; (2) one or more working tapes for slaves (besides their own working tapes for natural numbers); (3) a write-only output tape for Baire space output, the slaves will write sequentially, in particular, any slave cannot modify

³This approach is similar to the prime computability introduced by Moschovakis [10]. We thank Alexandra Soskova for informing us the literatures in this area.

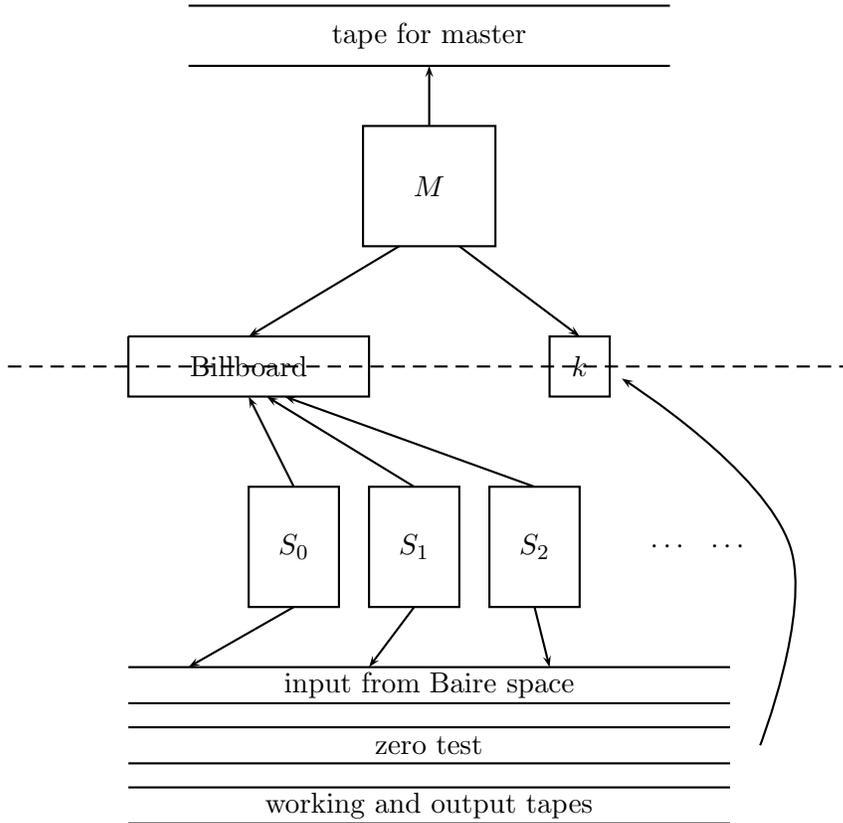


Figure 1: A Master-Slave machine

the writings of other slaves; (4) a special tape, called *zero-test tape*, for any sequence written on this tape, there is a special module which is able to detect if it is the zero sequence and to return a Boolean bit $k = 0$ or 1 to the master.

The interaction between the master and the slaves is done as follows. The master writes a natural number p on the *billboard* tape. Each slave S_i will take this p as its program input and its index i as numerical input. Since the slaves are universal oracle Turing machines Φ , what S_i does is to simulate the TTE-computation $\Phi^x(p, i)$. Here and below, we use $\Phi^x(p)$ for $\Psi(p; x)$ to emphasize that x is an oracle; and $\Phi^x(p, i)$ is the i -th component of $\Phi^x(p)$ if it is defined.) The feedback from the slaves to master is done via *zero-test* module, and the result is stored in the Boolean bit k which M can use.

The program (or software part) of a master-slave machine can be identified with the Turing program of the master M . We can ignore the slaves, because all slaves use the same fixed universal program. Fix a finite set of states $Q = Q_0 \cup \{S, E\}$ where Q_0 is the set of states for a standard Turing machine and S and E are two new symbols for slave calling and zero-testing respectively. We also single out two states q_s and q_h in Q_0 for starting and halting, respectively. Also fix a set of alphabet Σ , usually binary and with some auxiliary symbols for convenience. The transition functions of a master-slave machine can be represented by a finite set of finite tuples, the dimension of the tuples depends on the number of tapes. For example, the machine in Figure 2.2 has three tapes (and three heads), namely, input tape, billboard and the Boolean bit. However, in the discussion below, we will ignore the multi-tape issues for simplicity. We will use only one tape, hence the transition function is described by quadruples. Since any multi-tape standard Turing machine can be simulated by one with a single tape, we do not lose any generality here. The quadruples are of the following three kinds: Standard, slave-calling and zero-testing.

- (1) The standard commands are of the form $qaq'q'$ or $qaLq'$ or $qaRq'$, where $q \in Q_0$, $q' \in Q$, $a, a' \in \Sigma$ and L, R are for directions. Their executions are exactly the same as in classical case.
- (2) The slave calling commands are of the form $Saaq$, where $q \in Q$ and $a \in \Sigma$ (the symbol a is just to make it a quadruple). When executing this command, the slave S_i will perform $\Phi^x(p, i)$ as described in the interaction part above. When every slave machine halts, the state of the master becomes q . If some slave does not halt, the computation on input x of the MS-machine is undefined.
- (3) The zero-test commands are of the form $Eaaq$ where $q \in Q$ and the symbol a again is to make it a quadruple. The execution is also described in the interaction part above.

The definition of an MS-computation is analogous to the classical one. For simplicity again, we assume that we only have one tape for master and one tape for slaves. An *MS-configuration* C is a pair $(n; y)$ where n is a natural number coding the configuration (in the standard sense) of the master together with the billboard and Boolean bit; and y is an element in \mathcal{N} currently written on the tape handled by slaves.

Given a master-slave machine M and a configuration $C = (n; y)$, we decode from n in C to get the current state parameter q and the symbols a reading by the master; then check which quadruple in M starts with qa (in the case when q is S or E , we do not check the second component a) for some non-halting state q and act according to the command, the resulting configuration D will be the one *yielded* from C by M .

A configuration $(n; y)$ is called *terminal*, if its state component coded in n is the halting state q_h . A terminal configuration does not yield any configurations.

For any input $(k; x)$ of mixed type, the *initial MS-configuration* for input $(k; x)$ is the one that the Master is in its starting state q_s and k is put on the input tape of the master and the Baire space input x is written on the oracle tape shared by all slaves.

A *master-slave computation* of M on input $(k; x)$ is a sequence of MS configurations $\langle C_i \rangle$ such that C_0 is the initial configuration for $(k; x)$, and either (1) the sequence is infinite and for all i , C_i yields C_{i+1} ; or the sequence is of finite length $i_0 + 1$ for some natural number i_0 , for all $i < i_0$, C_i yields C_{i+1} , and either (2.1) C_{i_0} is a terminal MS-configuration, in which case, the output is the number on the output tape of the master or slaves depending on the codomain; or (2.2) the state component of C_{i_0} is S and $\Phi^y(p, j) \uparrow$ for some $j \in \omega$, where p and y are the billboard and Baire space component of C_{i_0} respectively.

Definition 2.11. We say that a partial function $f : \mathbb{N}^p \times \mathcal{N}^q \rightarrow Y$ where $Y \in \{\mathbb{N}, \mathcal{N}\}$ is *MS-computable* over \mathcal{N} if there is a master-slave machine M such that

$$f(\vec{n}; \vec{x}) = \begin{cases} y, & \text{if } M \text{ on input } (\vec{n}; \vec{x}) \text{ halts and the output is } y; \\ & \text{(note that } y \text{ can be in } \mathbb{N} \text{ or in } \mathcal{N} \text{ depending on } f.) \\ \uparrow, & \text{if (1) or (2.2) happens in the MS-computation of } M \\ & \text{on input } (\vec{n}; \vec{x}). \end{cases}$$

Lemma 2.12. *Every partial recursive function over \mathcal{N} is MS-computable over \mathcal{N} .*

Proof. By Definition 2.3, it suffices to check that the basic functions are MS-computable over \mathcal{N} ; and the class of MS-computable functions over \mathcal{N} are closed under composition, primitive recursion with respect to \mathbb{N} and μ -operator with respect to \mathbb{N} .

The basic functions in item 1(a) to 1(c) in Definition 2.3 are clearly MS-computable. The universal TTE-computable function is MS-computable: Given input $(e; x) \in \mathbb{N} \times \mathcal{N}$, the master just copies e on billboard and calls the slaves. The MS-computation of χ is simply done by the zero-test command.

The proof of the closure properties is similar to the ones in classical recursion theory, hence skipped. \square

2.3. A Normal Form Theorem for computation over \mathcal{N} . Our definition of MS-computation has several quantifiers over $\mathbb{N} \times \mathcal{N}$, it is not obvious how effective it is. In this subsection, we analyze MS-computations carefully and define an analog of Kleene's T -predicate over \mathcal{N} . Consequently we get both the following Normal Form Theorem and the main characterization of MS-computable functions.

Theorem 2.13 (Normal Form Theorem). *There are primitive recursive over \mathcal{N} predicate $T(e, x, z)$ and a (partial) TTE-computable function $U(z; x)$ where $e, z \in \mathbb{N}$ and $x \in \mathcal{N}$, such that for any MS-computable function F over \mathcal{N} , there is some e , for all $x \in \mathcal{N}$*

$$F(x) = U(\mu z T(e, x, z); x).$$

Theorem 2.14. *The class of MS-computable functions over \mathcal{N} coincides with the class of partial recursive functions over \mathcal{N} .*

We now proceed to prove Theorem 2.13. Let $F \in \mathcal{F}$ be MS-computable, say computed by an MS-machine M . For notational simplicity, we assume that F is from $\mathbb{N} \times \mathcal{N}$ to \mathcal{N} , the cases of $\mathbb{N}^p \times \mathcal{N}^q$ to \mathcal{N} or to \mathbb{N} are similar. And we will ignore the multi-tape issue again. Since there is an effective way to list all MS-programs, this M has an index, say e . This e will be carried along as a fixed parameter in the rest of the proof.

As in classical recursion theory, we need to “arithmetize” everything. The first important feature is that this arithmetization is done almost entirely using natural numbers, with minimal reference to the input x in Baire space. All transitions of configurations will be recorded by natural numbers only. To do that, we introduce a new natural number parameter c , called the *TTE component of a configuration*, such that for input x , the sequence $\langle \Phi^x(c, i) : i \in \omega \rangle$ is the current tape content written on the slave's tape (here we used the single tape convention). In other words, we code a configuration $(n; y)$ as $(n, c; x)$ where $y(i) = \Phi^x(c, i)$ for all $i \in \omega$. In particular, the initial configuration is coded as $(n_0, c_0; x)$ where c_0 is an index of the identity TTE function, namely, $\Phi^x(c_0, i) = x(i)$ for all $i \in \omega$;

and n_0 codes the starting state q_s and the natural number input k , etc. This idea is possible because the only change on the Baire space component is initiated by the slave calling command. When we execute that command we get the content of billboard tape p from n . By s - m - n theorem, there is a (standard) primitive recursive function $g(p, c)$ such that $\Phi^x(g(p, c)) = \Phi^{\Phi^x(c)}(p)$. In other words, all Baire space components that we ever see during the computation are TTE-images of the input x .

Introducing the TTE component is not enough because the master may call some partial TTE functions, whereas we want to make the Kleene T -predicate primitive recursive, in particular T must be total. The idea is to make full use the infinitely many slaves and delay the detection of the Π_2^0 question of totality until the last step.

We have our second important (though a bit artificial) feature. We add restrictions to the slave activities as follows. Add to the slave alphabet a new symbol, say $\#$, and restrict the running time of the i -th slave S_i to 2^i , in fact, the exact time bound is not important, as long as the i -th slave can finish reading relevant part of the tape and computing for i steps. If the computation task of S_i does not finish by step i , it writes a $\#$ and stop. The unfinished task is left to the next slave. Thus slave S_{i+1} would start from checking the computations of S_0, \dots, S_i , if it finds some unfinished computations, say S_j is the first slave whose task is unfinished, S_{i+1} will continue the task of S_j . If within its time restriction $i + 1$, S_{i+1} completed the task of S_j , it writes the result on tape and declare that S_j 's task is finished; otherwise, it writes $\#$ and stop.

Let $\mathcal{N}^\# = (\omega \cup \{\#\})^\omega$. For any element $x = (x(0), x(1), \dots) \in \mathcal{N}$, we say that $y \in \mathcal{N}^\#$ is a $\#$ -extension of x , if either

$$y = (\#^{n_0}, x(0), \#^{n_1}, x(1), \#^{n_2}, x(2), \dots)$$

for some $(n_i) \in \mathcal{N}$; or there is some $i \in \omega$ such that

$$y = (\#^{n_0}, x(0), \#^{n_1}, x(1), \dots, \#^{n_i}, x(i), \#, \#, \dots).$$

In the latter case, we say that y is a *divergent* $\#$ -extension with initial part $(x(0), \dots, x(i))$. We also let $\Phi_\#^x(e)$ to denote the ‘‘restricted’’ universal oracle Turing machine. Observe that $\lambda x. \Phi_\#^x(e) : \mathcal{N}^\# \rightarrow \mathcal{N}^\#$ is total for every $e \in \mathbb{N}$.

It should be noted that by changing alphabet, we can easily embed $\mathcal{N}^\#$ into \mathcal{N} effectively, thus continuously. Thus we could discuss everything within \mathcal{N} completely, the purpose of introducing $\mathcal{N}^\#$ is purely for intuitive clarity.

During the computation, the master will ‘‘live’’ in \mathcal{N} , whereas the restricted slaves work in $\mathcal{N}^\#$ (except the last step of converting output). For instance, suppose that the current configuration is $(n, c; x)$ and the master wants the slaves to computing $\Phi^y(p)$ where y is the current tape content $\Phi^x(c)$. However, the slaves now are working inside $\mathcal{N}^\#$, they typically see a $\#$ -extension $y^\#$ of y as input. The slaves then compute $\Phi_\#^{y^\#}(p^\#)$ where the program $p^\#$ will produce some $\#$ -extension of z , if $\Phi^y(p) \downarrow = z$. Furthermore, we can get $p^\#$ effectively by s - m - n Theorem. We summarized these facts in the lemma below:

Lemma 2.15. *There is a primitive recursive function $g^\#$ such that for any $x \in \mathcal{N}$, $p, c \in \omega$, we have*

- (a) *if $\Phi^x(c) \downarrow = y$, $\Phi^y(p) \downarrow = z$, then $\Phi_\#^x(g^\#(p, c))$ is a nondivergent $\#$ -extension of z ;*
- (b) *if $\Phi^x(c) \uparrow$ or $(\Phi^x(c) \downarrow = y$ but $\Phi^y(p) \uparrow)$, then $\Phi_\#^x(g^\#(p, c))$ is a divergent $\#$ -extension (with initial part being whatever $\Phi^{\Phi^x(c)}(p)$ produces).*

On the other hand, when the master asks for a zero-test of y , the slaves will execute some “extended zero-test” on some $\#$ -extension of y , which treats the $\#$ -symbols as zero. Thus if the original zero-test gives answer k , the extended test will also give the same answer.

Lemma 2.16. *There is a function $\chi_{\#} : \mathcal{N}^{\#} \rightarrow \{0, 1\}$, which is a composition of the zero-test function χ in \mathcal{N} and some TTE over $\mathcal{N}^{\#}$ function, such that, for each $y^{\#} \in \mathcal{N}^{\#}$ which is a $\#$ -extension of some $y \in \mathcal{N}$, $\chi_{\#}(y^{\#})$ gives the same answer as $\chi(y)$, more precisely,*

$$\chi_{\#}(y^{\#}) = \begin{cases} 1, & \text{if } y = 0_{\mathcal{N}} \text{ or } y^{\#} \text{ is divergent with initial part all zero;} \\ 0, & \text{if } y \neq 0_{\mathcal{N}} \text{ or } y^{\#} \text{ is divergent with non-zero initial part.} \end{cases}$$

With these two features in mind, the rest of the proof is parallel to the classical proof of Kleene’s Normal Form Theorem. Define the function $t((n, c; x), e) = ((m, d; x), e)$ reflecting the fact “ C yields D by M ” via their codes as follows:

First we decode from n to get the current state q and the current symbol read by the master, say a ; then check which command or quadruple in M (which is coded by e) starts with qa (in the case when q is S or E , we do not check the second component a). Depending on the command, we obtain the code of the resulting configuration D as follows.

- (1) If the quadruple is qaq' or $qaLq'$ or $qaRq'$, then the transition only affects the master part, say it changes n to m . Then in this case, $t((n, c; x), e) = ((m, c; x), e)$. Note that $n \mapsto m$ is primitive recursive over \mathcal{N} by Lemma 2.7.
- (2) If the quadruple is $Saaq$, where S is the slave calling state, then we can get the content of billboard tape p from n . Let $g^{\#}(p, c)$ be the function defined in Lemma 2.15. Set $t((n, c; x), e) = ((m, g^{\#}(p, c); x), e)$ and m is the resulting code of the configuration of the master part to record the change of state to q . Note that $(n, c) \mapsto (m, g^{\#}(p, c))$ is again primitive recursive over \mathcal{N} .
- (3) If the quadruple is $Eaaq$ where E is the special zero-test state, then the state of master changed from E to q and the boolean bit becomes $\chi_{\#}(\Phi_{\#}^x(c))$, where $\chi_{\#}$ are the function defined in Lemma 2.16. Let m be the new code of the master configuration. we have $t((n, c; x), e) = ((m, c; x), e)$. Note that $n \mapsto m$ is again primitive recursive over \mathcal{N} . Also note that it is this step that the input x is used.

Apply the definition by cases (Lemma 2.10) and by discussions above, the transition function t is primitive recursive over \mathcal{N} .

Clearly to determine whether a code $(n, c; x)$ of is one of a terminal configuration is primitive recursive: Just check if the state component q in n is the halting state q_h of M .

Define the *Kleene T-predicate* $T(e, x, z)$ by “ z is a (natural number) code of finite sequences $\langle (n_i, c_i) : i \leq |z| \rangle$ such that $(n_0, c_0; x)$ is the initial configuration and for all $i < |z|$, $t((n_i, c_i; x), e) = ((n_{i+1}, c_{i+1}; x), e)$ and $(n_{|z|}, c_{|z|}; x)$ is terminal”. Note: Although z is a natural number, we did not put z in front of x because we want to keep it the same form as in classical recursion theory.

Lemma 2.17. *$T(e, x, z)$ is primitive recursive over \mathcal{N} .*

Proof. It follows from that coding input, transition t and deciding terminal configuration are all primitive recursive over \mathcal{N} ; and primitive recursive predicates are closed under bounded quantification (Lemma 2.9). \square

Finally, by applying μ -operator, we can find the least z such that $T(e, x, z)$, if exists. (If the function has codomain \mathbb{N} , then the output reading function $u : \mathbb{N} \times \mathbb{N} \times \mathcal{N} \rightarrow \mathbb{N}$ is

exactly as in classical case.) We define the output reading functions $U : \mathbb{N} \times \mathcal{N} \rightarrow \mathcal{N}$ as follows: Given a code of a finite sequence of configurations z and the Baire space input x , we first get from z the set of all slave calling commands that M ever used, say the indices are p_1, \dots, p_r . From $i = 0$, we read through from $j = 1$ to r , the i -th non- $\#$ -component of $\Phi_{\#}^x(p_j)$. If any of the $\#$ -extension is divergent, then $U(z; x)$ is undefined, otherwise, we write the i -th non- $\#$ -component of $\Phi_{\#}^x(p_r)$ sequentially, that is the value of $U(z; x)$. Clearly, U is a partial TTE-computable function.

That finishes the proof of the Kleene Normal Form Theorem, consequently, we have the characterization theorem (Theorem 2.14).

2.4. Characterizing the computable/recursive sets. Recall that the basic open set in Baire space is of the form $[\sigma] = \{f \in \mathcal{N} : \sigma < f\}$ where $\sigma \in \omega^{<\omega}$. We use $\overline{[\sigma]}$ to denote the closed set $\mathcal{N} \setminus [\sigma]$.

Definition 2.18. A subset $O \subseteq \mathcal{N}$ is said to be *effectively open* or Σ_1^0 over \mathcal{N} if there is a (classically) recursive sequence $\langle \sigma_i \rangle_{i \in \omega}$ such that $O = \bigcup_i [\sigma_i]$. We say that $C \subseteq \mathcal{N}$ is *effectively closed* or Π_1^0 over \mathcal{N} if $\mathcal{N} \setminus C$ is effectively open.

A subset $A \subseteq \mathcal{N}$ is said to be an *effectively G_δ set* or Π_2^0 , if there is a recursive sequence $\langle \sigma_{i,j} \rangle$ such that $A = \bigcap_i \bigcup_j [\sigma_{i,j}]$. A is an *effectively F_σ set* or Σ_2^0 if its complement is effectively G_δ . A is Δ_2^0 over \mathcal{N} if it is both effectively G_δ and effectively F_σ .

Definition 2.19. We say that a subset $A \subseteq \mathcal{N}$ is *recursive over Baire space* if its characteristic function χ_A is partial recursive over \mathcal{N} .

Clearly, from its definition, recursive over \mathcal{N} sets are closed under complementation.

Lemma 2.20. *Every effectively open subset O of \mathcal{N} is recursive over \mathcal{N} , so is every effectively closed set.*

Proof. Let $O = \bigcup_i [\sigma_i]$ where $\langle \sigma_i \rangle_{i \in \omega}$ is a recursive sequence. We design an MS-machine M to compute O as follows: Given input $x \in \mathcal{N}$, the master of M writes the code of computing $\langle \sigma_i \rangle$ on the billboard; and asks the i -th slave S_i to check if $\sigma_i < x$, i.e., $x \in [\sigma_i]$; if so, S_i writes a 1 on the i -th cell of the zero-test tape; it writes a 0 otherwise. Next we use zero-test to get the boolean bit k . If x is in O then some slave will write a 1 on the zero-test tape, so the sequence being tested is non-zero, so $k = 0$; if x is not in O , then all slaves write 0 on the zero-test tape, so $k = 1$. In other words $x \in O$ if and only if $k = 0$. Note that M actually decides the membership of O in two master steps, one slave calling and one zero-test. \square

For example, the Cantor space 2^ω is recursive over \mathcal{N} as a subset of the Baire space.

Lemma 2.21. *If a subset A of Baire space is Δ_2^0 over \mathcal{N} , then A is recursive over \mathcal{N} .*

Proof. Since A is Δ_2^0 , both A and its complement are Σ_2^0 . Let $A = \bigcup_i \bigcap_j \overline{[\sigma_{i,j}]}$ and $\mathcal{N} \setminus A = \bigcup_i \bigcap_j \overline{[\tau_{i,j}]}$, where $\langle \sigma_{i,j} \rangle$ and $\langle \tau_{i,j} \rangle$ are recursive sequences in the classical sense. Suppose $x \in \mathcal{N}$ is the input for the MS-machine M which we are designing. Using the algorithm described in the proof of Lemma 2.20, M can decide if x is in the closed set $\bigcap_j \overline{[\sigma_{0,j}]}$ using two master steps. If x is in such a closed set, then M outputs 1 indicating that “ x is in A ”. Otherwise, M will try to decide if $x \in \bigcap_j \overline{[\tau_{0,j}]}$, if yes, output 0 indicating that “ x is in $\mathcal{N} \setminus A$ ”. Otherwise, try $\bigcap_j \overline{[\sigma_{1,j}]}$, then try $\bigcap_j \overline{[\tau_{1,j}]}$, etc. Since x is either in A or in $\mathcal{N} \setminus A$. This algorithm always halts and will give correct answer. \square

The converse of Lemma 2.21 is also true. The proof depends on another analysis of how MS-machine works, on top of the one given in the proof of the Normal Form Theorem.

Let M be an MS-machine. Firstly we make all slaves in M restricted as described in subsection 2.3, thus push the partialness issue to the very end. We associate to M a binary tree $\Gamma := \Gamma(M)$, called the *computation tree* for M , as follows.

Roughly speaking, Γ simply lists all possible M -computations in tree form. Each node σ in Γ is associated with two kinds of parameters: One is a (code of a) configuration as defined in the Normal Form Theorem; the other is a triple of indices $(i, j, k) = (i(\sigma), j(\sigma), k(\sigma))$, where i and j are indices for some effective Σ_1^0 and Π_1^0 sets O_i and C_j respectively; and k is index for the TTE function $\lambda x. \Phi_{\#}^x(k)$. It intuitively means that at σ , we restrict our attention only to the set $O_i \cap C_j$ which is determined by zero-tests, and $\Phi_{\#}^x(k)$ is applied to $x \in O_i \cap C_j$ which has produced the contents on the tape(s) for slaves.

The root of Γ is associated with the initial configuration with empty input, and is labelled (i_0, j_0, k_0) where i_0 and j_0 are indices of the whole space $\mathcal{N}^{\#}$, and $k_0 = c_0$ which is the index of identity function defined in the proof of Normal Form Theorem. Suppose that we have defined the associated configuration and the indices at σ . We can decode from the configuration to get the state component q . Depending on q we have the following possible cases:

Case 1. q is the halting state. Then declare σ is a terminal node on Γ .

Case 2. q is the slave calling state S . Then σ has a unique successor $\sigma^{\wedge}0$. We associate the ‘yielded’ configuration to $\sigma^{\wedge}0$. If the number written on the billboard for slave to execute is p , $k(\sigma^{\wedge}0) = g^{\#}(p, k(\sigma))$, where $g^{\#}$ is defined in Lemma 2.15. Set $i(\sigma^{\wedge}0) = i(\sigma)$, and $j(\sigma^{\wedge}0) = j(\sigma)$.

Case 3. q is the zero-test state E . Then σ has two successors $\sigma^{\wedge}0$ and $\sigma^{\wedge}1$. At node $\sigma^{\wedge}0$ (which indicates the test result is ‘No’, i.e., the sequence being tested has symbols other than zero and $\#$), we associate the ‘yielded’ configuration (whose boolean bit is 0 now) to $\sigma^{\wedge}0$. If the content of the zero-test is prepared via TTE-function $\Phi_{\#}(z)$, we can effectively find the index i^* for the Σ_1^0 -set $\Phi_{\#}(z)^{-1}[\mathcal{N}^{\#} \setminus \{0, \#\}^{\omega}]$, and index i^{**} for $O_{i(\sigma)} \cap O_{i^*}$. Set $i(\sigma^{\wedge}0) = i^{**}$. Set $j(\sigma^{\wedge}0) = j(\sigma)$ and $k(\sigma^{\wedge}0) = k(\sigma)$. Do the similar things at $\sigma^{\wedge}1$ (just replace O, i by C, j respectively).

Case 4. q is one of the standard states. Then σ has a unique successor $\sigma^{\wedge}0$. Read from the configuration associated at σ the current command qaq' and the current scanned symbol b . At node $\sigma^{\wedge}0$ we associate the corresponding ‘yielded’ configuration to $\sigma^{\wedge}0$. Leave all indices unchanged.

This finishes the construction of the computation tree Γ . Clearly Γ is a recursive tree with $\sigma \mapsto (i(\sigma), j(\sigma), k(\sigma))$ recursive.

From this computation tree, we can get several consequences.

Theorem 2.22. *A partial function $F : \mathcal{N} \rightarrow \mathcal{N}$ is MS-computable if and only if there are a (classical) recursive set $R \subseteq \omega$ and a recursive function $\alpha : R \rightarrow \omega^3$ mapping $\sigma \mapsto (i, j, k')$ satisfying the following conditions:*

- (a) *The sets $O_{i(\sigma)} \cap C_{j(\sigma)}$ are mutually disjoint for $\sigma \in R$.*
- (b) *If $F(x)$ is defined, then there is a $\sigma \in R$ such that $x \in O_{i(\sigma)} \cap C_{j(\sigma)}$ and $F(x) = \Phi^x(k')$.*
- (c) *If $F(x)$ is undefined, then either $x \notin \bigcup_{\sigma \in R} (O_{i(\sigma)} \cap C_{j(\sigma)})$ or $x \in O_{i(\sigma)} \cap C_{j(\sigma)}$ for some $\sigma \in R$ but $\Phi^x(k')$ is undefined.*

Proof. Suppose that F is computed by the MS-machine M , let Γ be its computation tree described above. Define $R = \{\sigma \in \Gamma : \text{the state component of the configuration associated}$

with σ is halting}. (Here we identify the node σ on Γ with its Gödel number.) Since Γ is recursive, so is R . Let α map σ to (i, j, k') , where (i, j, k) is the label of σ in Γ and k' is the index of the TTE computable function $U \circ \lambda x. \Phi_{\#}^x(k)$, where U is the output reading function defined in the Normal Form Theorem. To see that (a) is satisfied, any branching at Γ can only be caused by Case 3 in the definition of Γ . The split is caused by zero-test and the resulting sets are certainly disjoint.

For (b) if $F(x)$ is defined, we will hit a halting node σ on Γ , so $\sigma \in R$ and all slave calls along the computation path up to σ are not partial, so $\Phi^x(k)$ would produce the correct result $F(x)$. Similarly (c) holds.

Conversely, suppose that we have recursive R and α , we design an MS-machine M as follows. For any input x , M checks through every element $\sigma \in R$ to see if $x \in O_{i(\sigma)} \cap C_{j(\sigma)}$. For each fixed σ , this can be done with finitely many master steps as in the proof of Lemma 2.20. If no such σ is found, then $M(x) \uparrow$, however by (b), $F(x)$ also diverges. If such σ is found, then M just call the TTE functional $\Phi^x(k')$, by (b) and (c), $M(x) = F(x)$. \square

Corollary 2.23. *The domain W of an MS-computable function is of the form $\bigcup_n [(O_{i_n} \cap C_{j_n}) \cap Y_n]$, where Y_n is Π_2^0 subset $\{x : \Phi^x(k_n) \in \text{domain of } U\}$, and $n \mapsto (i_n, j_n, k_n)$ is primitive recursive.*

Theorem 2.24. *If a subset A of Baire space is recursive over \mathcal{N} , then A is Δ_2^0 over \mathcal{N} .*

Proof. Let M be an MS-machine computing the characteristic function χ_A and R be the recursive set defined in Theorem 2.22. Since M is total, for every $\sigma \in R$, the domain of $\lambda x. \Phi^x(k')$ has to be a superset of $O_{i(\sigma)} \cap C_{j(\sigma)}$, where k' is also as defined in Theorem 2.22. Thus we have $x \in A$ if and only if for some $\sigma \in R$ with $x \in O_{i(\sigma)} \cap C_{j(\sigma)}$ and the natural number output read from the configuration associated with σ is 1. Consequently, A is Σ_2^0 . Similarly the complement of A is also Σ_2^0 . So A is Δ_2^0 over \mathcal{N} . \square

This result shows that recursive sets are closer to the intuitively computable sets. For instance, they do not include all Borel sets, in fact, not all arithmetical sets.

For subsets of natural numbers, we have a similar characterization.

Theorem 2.25. *A subset A of natural numbers is MS-computable if and only if $A \leq_T \emptyset'$, i.e., it is a Δ_2^0 -subset of natural numbers.*

Proof. Observe that the natural number input is directly sent to the master, whereas the input tape for slaves are empty at the beginning of the computation.

Suppose that A is MS-computable, say by the MS-machine M . For any input n , we trace the computation tree Γ associated with M . At each branching point σ cause by the zero-test, since the content prepared for the zero test is by $\Phi^{\emptyset}(k(\sigma))$ where k is defined in Theorem 2.22. Furthermore, since χ_A is total, $\Phi^{\emptyset}(k(\sigma))$ is total. Hence whether the sequence is the zero-sequence is a Π_1^0 question:

$$\forall n \forall s (\Phi^{\emptyset}(k(\sigma), n) \downarrow [s] \rightarrow \Phi(k(\sigma), n) = 0).$$

So \emptyset' can produce the answer. Consequently, \emptyset' can trace Γ until the terminal node and read off the answer of $A(n)$. So $A \leq_T \emptyset'$.

On the other hand, if $A \leq_T \emptyset'$ say via the oracle Turing machine $\Phi^{\emptyset'}(e)$. The master can mimic $\Phi(e)$ until it reaches a query step asking if some number n is in \emptyset' . At that moment, the master employs the slaves to figure it out and read the answer from the boolean bit. Thus, A is MS-computable. \square

2.5. Relation with BSS models. In [1], L. Blum, Shub and Smale introduced a highly influential model (BSS machines) of computation over real numbers, in fact, over arbitrary rings. BSS machines treat a real number or an element in Baire space as a single mathematical entity, and only focus on functions which are mostly common in scientific computing, such as polynomials or rational functions. To compare our model with BSS machines, we have to make some reasonable modifications to both models. First of all, BSS machines can use arbitrary real numbers as parameters, in particular, there are uncountably many BSS machines. We have to restrict BSS model so that it does not use real parameters.

Now let us discuss computability on (ordered) rings with three basic ring operations $+$, $-$ and \times . We modify MS-machines as follows:

- (1) We assume that we have only three slaves to compute the basic ring operations respectively. Thus by calling them finitely many times, we can compute any polynomial in $\mathbb{Z}[\vec{x}]$.
- (2) We assume the zero-test module can tell whether a nonzero element is positive or negative too.

With these modification, we have

Theorem 2.26. *Over a ring R , the modified MS-machines compute exactly the same class of functions as the (parameter free) BSS-machines.*

Proof. (Sketch) Define the class of *partial recursive functions* over an ordered ring R as the smallest class \mathcal{C} of functions on $\mathbb{N}^p \times R^q$ satisfying the following conditions:

- (1) \mathcal{C} contains the following basic functions:
 - (a) Zero function $Z : \mathbb{N} \rightarrow \mathbb{N}$ with $Z(n) = 0$;
 - (b) successor function $S : \mathbb{N} \rightarrow \mathbb{N}$ with $S(n) = n + 1$; and
 - (c) for natural numbers p, q and i with $p + q \geq 1$ and $1 \leq i \leq p + q$ the projection function

$$\pi_i^{p+q}(n_1, \dots, n_p; x_1, \dots, x_q) = \begin{cases} n_i, & \text{if } i \leq p; \\ x_{i-p}, & \text{if } i > p. \end{cases}$$

- (d) The primitive functions in ring R , namely $+_R$, $-_R$ and \times_R ; and
- (e) The sign function $\text{sgn} : R \rightarrow \mathbb{N}$ defined by

$$\text{sgn}(x) = \begin{cases} 1, & \text{if } x > 0_R; \\ 0, & \text{if } x = 0_R; \\ 2, & \text{if } x < 0_R. \end{cases}$$

- (2) \mathcal{C} is closed under
 - (a) composition, provided the types are matched;
 - (b) primitive recursion with respect to \mathbb{N} ; and
 - (c) μ -operator with respect to \mathbb{N} .

By mimic the proof of the Normal Form Theorem, one can show that the modified MS-machines compute exactly the partial recursive functions over R . On the other hand, by Theorem on page 33 in [1] section 7, BSS machines compute exactly the same class. \square

On the other hand, let us look at computability on the Baire space. First of all, we delete from BSS model the function of taking additive inverse, as we focus on functions on \mathcal{N} , instead of on rings. Since BSS machines as defined originally does include functions beyond

polynomials, for example, certain sets⁴ involving exponential function is not computable in BSS sense. It is necessary to add functions that we are interested in, for instance, exponential function. Obviously, we do not want to add them “in the hardware” i.e., merely allow those functions to appear in the computing nodes of BSS machines, otherwise, we merely move up one level of primitive recursive hierarchy and we will miss, say, tower of exponential functions. Instead, we should add them through universal functions, so that the machine can call them dynamically. Thus, we add the universal TTE-computable function $\Phi(e; x)$ in the BSS model, so that it can call any TTE-computable function.

Theorem 2.27. *Over the Baire space \mathcal{N} , the modified BSS-machines compute exactly the same class of functions as the MS-machines.*

Proof. (Sketch) By slightly modifying the proof of Theorem on page 33 in [1] section 7, modified BSS-machines can compute all partial recursive functions over \mathcal{N} . On the other hand, Turing machines can mimic all computations defined by flow-charts, thus MS-machines can compute all functions that modified BSS-machine can compute. The theorem then follows from Theorem 2.14. \square

Theorem 2.27 tells us (informally) that over \mathcal{N} , MS-computation model is the smallest one that includes both BSS and TTE computation. The reasons are: MS-machine is certainly more powerful than BSS and TTE. On the other hand, if a computation model includes both BSS and TTE, then it would be more powerful than the modified BSS machine. By Theorem 2.27, it is more powerful than MS-machine.

3. COMPUTABILITY THEORY ON REAL NUMBERS

3.1. Formalize Computability over Real Numbers. We now discuss the computability over real numbers \mathbb{R} . Although Baire space is often viewed as a representation of \mathbb{R} , computationally and topologically, \mathcal{N} is quite different from “the real numbers” that “working mathematicians”, like numerical analysts, have been working on. We will make our meaning precise in the discussion below.

First of all, by well-known effective codings, we can identify the sets of integers \mathbb{Z} and rational numbers \mathbb{Q} with the set of natural numbers \mathbb{N} , respectively. In particular, we will not study negative numbers explicitly. Again by effective coding, we can identify the arithmetical operations of the field \mathbb{Q} , the ordering on \mathbb{Q} and functions like absolute value function, as primitive recursive functions over \mathbb{N} . Through this coding, infinite rational sequences will be identified with infinite sequences of natural numbers. We use \mathbb{Q}^ω to denote the set of all infinite rational sequences. It is worth noting that the coding is not a homeomorphism between the Baire space and \mathbb{Q}^ω equipped with the topology induced by the usual metric on \mathbb{Q} . Therefore, we will use \mathbb{Q}^ω instead of \mathcal{N} .

We fix the following presentation of real numbers:

Definition 3.1. (1) A *fast converging Cauchy sequence* is a sequence of rational numbers $\langle r_i : i \in \omega \rangle$ such that for all $m < n$ $|r_m - r_n| < 2^{-m-1}$. Remark: In the remaining of the paper, since we will focus only on fast converging Cauchy sequences, whenever we say Cauchy sequences we mean fast converging ones.

⁴There are examples over real numbers (see Brattka [2]); but can be adapted over Baire spaces.

- (2) Define a relation \sim between Cauchy sequences by $\langle r_i \rangle \sim \langle s_i \rangle$ if for any n there is some $m > n$ such that $|r_m - s_m| < 2^{-n}$ (hence $\lim_i r_i = \lim_i s_i$).
- (3) Observe that \sim defined in (2) is an equivalence relation. We identify the \sim -equivalence classes as real numbers. The set of real numbers is denoted by \mathbb{R} .

In this section we use i, j, k to denote natural numbers, α, β to denote infinite Cauchy sequences, and x, y to denote real numbers.

A (finite) rational sequence $\sigma = \langle q_0, \dots, q_{n-1} \rangle \in \mathbb{Q}^{<\omega}$ is said to be *Cauchy* if for every $i < j < n$ we have $|q_i - q_j| < 2^{-i-1}$. Given $\sigma = \langle q_0, \dots, q_{n-1} \rangle, \tau = \langle r_0, \dots, r_{m-1} \rangle \in \mathbb{Q}^{<\omega}$, we say that σ and τ are *Cauchy compatible* or simply *C-compatible* if $|q_{n-1} - r_{m-1}| < 2^{-n} + 2^{-m}$. The point of C-compatibles strings is that they can be extended to \sim -equivalent infinite Cauchy sequences; whereas the non C-compatible ones cannot. In other words, if $\alpha \supset \sigma$ and $\beta \supset \tau$ and $\alpha, \beta \in \mathbb{Q}^\omega$ and σ and τ are not C-compatible, then $\lim \alpha \neq \lim \beta$; equivalently, if $\lim \alpha = \lim \beta$ then for every i, j we have that $\alpha \upharpoonright i$ and $\beta \upharpoonright j$ are C-compatible.

We say that a function $F : \mathbb{Q}^\omega \rightarrow \mathbb{Q}^\omega$ is *Cauchy preserving*⁵ if F is only defined on Cauchy sequences and if two Cauchy sequences α and β are \sim -equivalent and $F(\alpha) \downarrow$ then $F(\alpha)$ is a Cauchy sequence, $F(\beta) \downarrow$ is also a Cauchy sequence and $F(\alpha) \sim F(\beta)$. Clearly, each Cauchy preserving function $F : \mathbb{Q}^\omega \rightarrow \mathbb{Q}^\omega$ naturally induces a function \tilde{F} from \mathbb{R} to \mathbb{R} .

Definition 3.2. We say that a function $\tilde{F} : \mathbb{R} \rightarrow \mathbb{R}$ is *TTE-computable* over \mathbb{R} , if there exists a TTE-computable function $F : \mathbb{Q}^\omega \rightarrow \mathbb{Q}^\omega$ which is Cauchy preserving and \tilde{F} is induced by F .

This definition suggests the two-step approach towards computability on \mathbb{R} , the first is to have an algorithm working on a representative α , and the second is to check that this algorithm is Cauchy preserving. The second step is the difference between computability over \mathcal{N} and over \mathbb{R} . This is more pertinent when we talk about machines. Over \mathcal{N} , the input of the TTE-computable functions and the object written on the input tape of a machine are the same; whereas over \mathbb{R} , the input of the functions are equivalence classes which cannot be written on the input tape. A typical oracle Turing machine is sensitive to the digits on the oracle tape, hence will not be Cauchy preserving at all. The main issue is to “fine-tune” the machine so that its operations are independent of the representation of the equivalence class. Once the fine-tuning is done, the rest is similar to Section 2.

We say that an oracle Turing machine \widehat{M} is *fine-tuned* if it satisfies the following two conditions: (1) If $\alpha \in \mathbb{Q}^\omega$ is not Cauchy, then \widehat{M}^α is undefined; (2) if α and β are \sim -equivalent Cauchy sequences, then \widehat{M}^α is defined if and only if \widehat{M}^β is defined and the outputs \widehat{M}^α and \widehat{M}^β are also \sim -equivalent Cauchy sequences.

We describe an effective procedure which transfers an oracle Turing machine M into a fine-tuned one. By this transfer, the universal oracle Turing machine will become a fine-tuned universal oracle Turing machine, which naturally induces a universal TTE-function over \mathbb{R} .

The plan is to associate a “canonical” object to a real number x , which is invariant under its representations. One candidate for the canonical object is “the” binary expansion of x . Unfortunately, it is not effective to extract a unique binary expansion for x , for

⁵This and other notions in this section are studied extensively by the school of computable analysis, see Chapter 3 in [18]. That is why we use the phrase “TTE-computable”. However, since we only study the computability on \mathbb{R} , we did not introduce the general naming systems, admissibility etc. Our approach and theirs are essentially similar, but may be different in some details.

example, when x is a dyadic rational, there are two such expansions. Recall that a dyadic rational is a rational number whose denominator is a power of 2. To overcome this problem, for a given Cauchy sequence α , we effectively produce a binary tree T_α whose infinite paths are exactly the binary expansion(s) of $x = \lim \alpha$; namely, if x is not a dyadic rational, then T_α has a unique infinite path which is the binary expansion of x ; if x is dyadic then T_α has exactly two paths which are the two binary expansions of x . If α and β are Cauchy equivalent, then T_α and T_β may not be the same, but they will have the same set of infinite paths.

The algorithm that transfer a Cauchy sequence α to T_α goes as follows. For each natural number n , let D_n denote the dyadic rational number with denominator 2^n , i.e. $D_n = \{\frac{m}{2^n} : m \in \mathbb{Z}\}$ and $D = \bigcup_{n \in \mathbb{N}} D_n$ be the set of all dyadic rational numbers. For natural number n and integer m , let

$$J_n^m = \left[\frac{m}{2^n}, \frac{m+1}{2^n} \right]$$

which is a closed interval of length $\frac{1}{2^n}$. Given a Cauchy sequence α , let

$$I_n(\alpha) = \left[\alpha(n+4) - \frac{1}{2^{n+2}}, \alpha(n+4) + \frac{1}{2^{n+2}} \right],$$

so that $I_n(\alpha)$ has the following properties:

- (1) $I_n(\alpha)$ is a closed interval of length $\frac{1}{2^{n+1}}$.
- (2) Since α is Cauchy, $x = \lim \alpha$ is in $I_n(\alpha)$.
- (3) $I_{n+1}(\alpha) \subseteq I_n(\alpha)$.

We now define the binary tree T_α in terms of $I_n := I_n(\alpha)$ and J_n^m . Let \emptyset be the root of T_α . The recursive construction of T_α and the labelling process are as follows. Since I_0 is of length $\frac{1}{2}$, it can intersect at most two intervals of the form J_0^m for some m . If I_0 intersects two such intervals, say $J_0^{m_0}$ and $J_0^{m_0+1}$ for some $m_0 \in \mathbb{Z}$, then the root has two children labelled from left to right as $d_L^0 = \frac{m_0}{2^0}$ and $d_R^0 = \frac{m_0+1}{2^0}$ respectively; if I_0 intersects only one such interval, say $J_0^{m_0}$ for some $m_0 \in \mathbb{Z}$, then the root has only one child labelled $d^0 = \frac{m_0}{2^0}$.

Suppose that we have defined the tree T_α up to level $\leq \ell$. First check if the finite sequence $\alpha \upharpoonright (\ell+4)$ is Cauchy, if it is not, then stop the construction; otherwise, proceed as follows. If T_α has a single node of length ℓ , then proceed as the base case, except replacing 0 by $\ell+1$. If T_α has two nodes of length ℓ , say ρ_L and ρ_R , labelled by dyadic rationals $d_L^\ell, d_R^\ell \in D_\ell$ respectively, where $d_R^\ell = d_L^\ell + \frac{1}{2^\ell}$. Find the interval $I_{\ell+1}$. Since it has length $\frac{1}{2^{\ell+2}}$, it can intersect at most two intervals of the form $J_{\ell+1}^m$ for some m .

Case 1. $I_{\ell+1}$ intersects two such J -intervals. Since the $I_{\ell+1} \subseteq I_\ell$, the J -intervals have to be $[d_L^\ell + \frac{1}{2^{\ell+1}}, d_R^\ell]$ and $[d_R^\ell, d_R^\ell + \frac{1}{2^{\ell+1}}]$. We extend ρ_L to a single node labelled $d_L^\ell + \frac{1}{2^{\ell+1}}$ and ρ_R to a single node labelled d_R^ℓ .

Case 2. $I_{\ell+1}$ intersects only one such J -interval. Again since the I -intervals are nested, the J -interval has to be either $[d_L^\ell + \frac{1}{2^{\ell+1}}, d_R^\ell]$ or $[d_R^\ell, d_R^\ell + \frac{1}{2^{\ell+1}}]$. If it is former, extend ρ_L to a single node labelled by $d_L^\ell + \frac{1}{2^{\ell+1}}$ and declare ρ_R a dead end; if it is latter, extend ρ_R to a single node labelled by d_R^ℓ and declare ρ_L a dead end. That finished the construction of the tree T_α .

By construction, T_α has the following properties:

- (1) T_α is an infinite binary tree if and only if α is Cauchy.

- (2) Assuming that α is Cauchy and $x = \lim \alpha \notin D$, then T_α has a unique path $\delta = \delta(T_\alpha)$ such that for each ℓ , $\delta(\ell)$ is the largest dyadic in D_ℓ which is less than x .
- (3) Assuming that α is Cauchy and $x = \lim \alpha \in D_n$ for the least such n , then for all $\ell \geq n$, $\delta_R(T_\alpha)(\ell) = x$ and $\delta_L(T_\alpha)(\ell) = x - \frac{1}{2^\ell}$.
- (4) The function from $\alpha \upharpoonright (\ell + 4) \mapsto T_\alpha \upharpoonright \ell$ is recursive.

It follows from (2) and (3) that if two Cauchy sequences α and β are equivalent then T_α and T_β have the same paths.

With the help of T_α , we can describe an effective procedure which transfers an oracle Turing machine M into a fine-tuned \widehat{M} . The basic idea is that we monitor the (at most two) infinite paths δ_L and δ_R of T_α , \widehat{M} simulates M^{δ_L} and M^{δ_R} and we accept the value only when the results are C-compatible. The details are as follows:

Given M , n and α , the fine-tuned machine \widehat{M} first generates the tree T_α (by property (4) above, this can be done), and simulates $M^{\delta_L}(n)$ and $M^{\delta_R}(n)$, where δ_L and δ_R are the paths through T_α (they might be equal) until one of the following happens:

- (a) There is a natural number s , for all $i \leq n$, $M^{\delta_L \upharpoonright s}(i) \downarrow = r_i$ and $\langle r_i : i \leq n \rangle$ is a finite Cauchy sequence and $\delta_L \upharpoonright s$ is the only node of length s that still have extensions on T_α . Set the output to be r_n .
- (b) There is a natural number s , for all $i \leq n$, $M^{\delta_L \upharpoonright s}(i) \downarrow = r_i$, $M^{\delta_R \upharpoonright s}(i) \downarrow = t_i$ and $\langle r_i : i \leq n \rangle, \langle t_i : i \leq n \rangle$ are finite Cauchy sequences, $|r_i - t_i| \leq \frac{1}{2^i}$ and both $\delta_L \upharpoonright s$ and $\delta_R \upharpoonright s$ still have extensions on T_α . Set the output to be $\frac{r_n + t_n}{2}$.

In case no such s exists, the procedure produces no output for this set of input M, n, α .

Observe that if $\widehat{M}^\alpha(n) \downarrow = a_n$ for every n , then $\langle a_n : n \in \mathbb{N} \rangle$ is a Cauchy sequence. When $M^\alpha = \beta$ is a Cauchy sequence with $y = \lim \beta$, $|a_n - \beta(n)| < \frac{1}{2^n}$. Let $b_n = a_{n+2}$ we will have

$$|b_n - y| = |a_{n+2} - y| \leq |a_{n+2} - \beta(n+2)| + |\beta(n+2) - y| \leq \frac{1}{2^{n+2}} + \frac{1}{2^{n+2}} < \frac{1}{2^n}.$$

Consequently, \widehat{M}^α and M^α are Cauchy equivalent. Finally the procedure of getting \widehat{M} from M is effective.

We summarize the discussion as follows:

Lemma 3.3. *Let $\langle \lambda\alpha.\Phi^\alpha(e) : e \in \omega \rangle$ be an effective list of all TTE computable functions over \mathbb{Q}^ω . Then there is a total recursive function $g : \mathbb{N} \rightarrow \mathbb{N}$ such that $\lambda\alpha.\Phi^\alpha(g(e))$ is Cauchy preserving. Furthermore, if $\lambda\alpha.\Phi^\alpha(e)$ is Cauchy preserving itself, then $\lambda\alpha.\Phi^\alpha(g(e))$ and $\lambda\alpha.\Phi^\alpha(e)$ induce that same function from \mathbb{R} to \mathbb{R} .*

We verify the expected fact⁶ for TTE-computable functions with respect to the usual topology:

Lemma 3.4. *Any TTE-computable function f over \mathbb{R} is continuous. To be more precise, suppose that $f : \mathbb{R} \rightarrow \mathbb{R}$ is computed by the fine-tuned MS-machine M , and $f(x_0) \downarrow = y_0$, then for any $n > 0$, there is an $m > 0$ such that for any Cauchy sequences α with $\lim \alpha = x_0$ and for any Cauchy sequence β with $\lim \beta = x$, if $M^\beta \downarrow$ and $|x_0 - x| < \frac{1}{m}$, then $|M^\beta - y_0| < \frac{1}{n}$.*

⁶Various versions of Lemmas 3.4 and 3.12 are well-known for computable analysts. For example, Lemma 3.4 may follow from Theorem 4.3.1 in Weihrauch [18] and passing through effective quotient topology. For the sake of completeness and correctness under our setting, we include outlines of the proofs.

Proof. Given $\frac{1}{n} > 0$, choose i such that $\frac{1}{2^i} < \frac{1}{2n}$. Since $f(x_0) \downarrow$, for any α with limit x_0 , $M^\alpha \downarrow$. Choose j such that $M^{\alpha \uparrow j}(i) \downarrow$, by definition, $|M^{\alpha \uparrow j}(i) - y_0| < \frac{1}{2^i}$. Now, by the fine-tuning, M will first produce T_α . At the stage s when $M^{\alpha \uparrow j}(i)[s] \downarrow$, there are two possible cases. Case 1. T_α has a unique node of length j that is still having extensions on T_α at stage s . By fine-tuning, the closed interval $I_j(\alpha) = [a, b]$ is a proper subinterval of some dyadic interval $J_j^{m_0} = [c, d]$, thus $c < a < b < d$. Let m be such that $\frac{1}{m} < \min\{\frac{a-c}{2}, \frac{d-b}{2}\}$. Case 2. T_α has a two nodes of length j that are still having extensions on T_α at stage s . By fine-tuning, the closed interval $I_j(\alpha) = [a, b]$ intersects two consecutive dyadic intervals $J_j^{m_0} = [c, c']$ and $J_j^{m_0+1} = [c', d]$, thus $c < a \leq c' \leq b < d$. Let m be such that $\frac{1}{m} < \min\{\frac{a-c}{2}, \frac{d-b}{2}\}$. In both cases, if a Cauchy sequence β satisfying $|x_0 - \lim \beta| < \frac{1}{m}$, the node(s) of length j that is still having extensions on T_β are also on T_α . Thus $M^{\beta \uparrow j}(i) = M^{\alpha \uparrow j}(i)$. If $M^\beta \downarrow$, then it is a Cauchy sequence, and

$$|M^\beta - y_0| < |M^\beta - M^{\beta \uparrow j}(i)| + |M^{\beta \uparrow j}(i) - y_0| < \frac{1}{2^i} + |M^{\alpha \uparrow j}(i) - y_0| < \frac{1}{n}.$$

□

Theorem 3.5 (Enumeration Theorem for TTE-computable functions). *There is a universal function $\Psi(e; x) : \mathbb{N} \times \mathbb{R} \rightarrow \mathbb{R}$ for TTE-computable over \mathbb{R} , i.e., for any TTE-computable function $\tilde{F} : \mathbb{R} \rightarrow \mathbb{R}$, there is some $e \in \mathbb{N}$ such that for all $x \in \mathbb{R}$, $\tilde{F}(x) = \Psi(e; x)$.*

Definition 3.6. The class of *partial recursive functions over \mathbb{R}* is the smallest subclass \mathcal{C} of \mathcal{F} satisfying the following conditions:

- (1) \mathcal{C} contains the following basic functions:
 - (a) Zero function $Z : \mathbb{N} \rightarrow \mathbb{N}$, $Z(n) = 0_{\mathbb{N}}$;
 - (b) successor function $S : \mathbb{N} \rightarrow \mathbb{N}$, $S(n) = n + 1$; and
 - (c) for natural numbers p, q and i with $p + q \geq 1$ and $1 \leq i \leq p + q$ the projection function
$$\pi_i^{p+q}(n_1, \dots, n_p; x_1, \dots, x_q) = \begin{cases} n_i, & \text{if } i \leq p; \\ x_{i-p}, & \text{if } i > p. \end{cases}$$
 - (d) A universal TTE-computable functions $\Psi(e; x)$ over \mathbb{R} ; and
 - (e) the characteristic function $\chi : \mathbb{R} \rightarrow \mathbb{N}$ of $\{0_{\mathbb{R}}\}$ where $0_{\mathbb{R}}$ is the real number zero.
- (2) \mathcal{C} is closed under
 - (a) composition, provided the types are matched;
 - (b) primitive recursion with respect to \mathbb{N} ; and
 - (c) μ -operator with respect to \mathbb{N} .

We say that an MS-machine is fine-tuned, if it only writes index $g(e)$ on its billboard where g is the function defined in Lemma 3.3.

Definition 3.7. We say that a partial function $f : \mathbb{N}^p \times \mathbb{R}^q \rightarrow \mathbb{R}$ is *MS-computable* over \mathbb{R} or simply MS-computable if there is a fine-tuned master-slave machine M such that

$$f(\vec{n}; \vec{x}) = \begin{cases} y, & \text{if for any representation } (\vec{n}; \vec{\alpha}) \text{ of input,} \\ & M^{\vec{\alpha}}(\vec{n}) \downarrow = \beta \text{ and } \beta \text{ is a representation of } y; \\ \text{undefined,} & \text{if for any representation } (\vec{n}; \vec{\alpha}) \text{ of input,} \\ & M^{\vec{\alpha}}(\vec{n}) \text{ never halts; or} \\ & \text{some slave calls are partial during} \\ & \text{the computation.} \end{cases}$$

We define $f : \mathbb{N}^p \times \mathbb{R}^q \rightarrow \mathbb{N}$ being *MS-computable* similarly.

Lemma 3.8. *The characteristic function of $\{0_{\mathbb{R}}\}$ is MS-computable over \mathbb{R} .*

Proof. This is because all Cauchy sequences are fast converging. Given α which is any presentation of x , we may ask the slaves S_i to check if

$$\forall j < i (|\alpha(j)| < \frac{1}{2^{j-1}})$$

and write a zero if the answer is yes, write $\frac{\alpha(i)}{2}$ otherwise. Observe that this preparation function is TTE and Cauchy preserving. Then the usual zero-test would determine if $x = 0_{\mathbb{R}}$. \square

3.2. Properties of Computable Sets and Functions over \mathbb{R} . We will establish some basic results on computability over \mathbb{R} , many of which have corresponding ones over the Baire space \mathcal{N} . Although the statements look similar, we actually cannot directly transform the results from \mathcal{N} to \mathbb{R} , because of the step 2 mentioned in the remarks after Definition 3.2. For example, the maps

$$((x(0), x(1), \dots), (y(0), y(1), \dots)) \mapsto (x(0), y(0), x(1), y(1), \dots)$$

is continuous from \mathcal{N} to \mathcal{N} , but the corresponding map from $\mathbb{Q}^{\omega} \times \mathbb{Q}^{\omega} \rightarrow \mathbb{Q}^{\omega}$ is not Cauchy preserving.

Also note that since BSS machines completely ignore the issue of representations, we will not compare with BSS models over \mathbb{R} .

3.2.1. Normal Form Theorem. As in section 2, we have the following Normal Form Theorem and characterization theorem over \mathbb{R} .

Theorem 3.9 (Normal Form Theorem for \mathbb{R}). *There are TTE computable over \mathbb{R} function $U(z; x)$ and a primitive recursive predicate $T(e, x, z)$ over \mathbb{R} such that for any partial recursive function F over \mathbb{R} , there is some e , for all $x \in \mathbb{R}$*

$$F(x) = U(\mu z T(e, x, z), x).$$

Theorem 3.10. *The class of MS-computable functions over \mathbb{R} coincides with the class of partial recursive functions over \mathbb{R} .*

As the proofs are similar to the one in Baire space, we only check the issues related to Cauchy preserving step. When executing the slave calling command, the function $g^\#(p, c)$ does not depend on the real input x . When executing the zero-test command, the preparation step, i.e., writing the sequence being tested on zero-test tape, is done by “fine-tuned” slaves, hence is Cauchy preserving (see the proof of Lemma 3.8). For the issue of working in $\mathcal{N}^\#$, one can do the same for the space \mathbb{Q}^ω with the extra symbol $\#$. For instance, one can define a sequence (finite or infinite) with $\#$ symbols $\#$ -Cauchy if it is Cauchy after removing the $\#$ symbols, and fine-tune the machines to make them “ $\#$ -Cauchy preserving”, instead of “Cauchy preserving”. The same proof will go through.

3.2.2. Characterizing the computable sets over \mathbb{R} . Recall that the basic open set in \mathbb{R} is of the form $B(c; r) = \{x \in \mathbb{R} : |x - c| < r\}$ where $c, r \in \mathbb{Q}$ and $r > 0$.

Definition 3.11. A subset $A \subseteq \mathbb{R}$ is said to be *effectively open* or Σ_1^0 over \mathbb{R} if there is a (classical) computable sequence $\langle e_i \rangle_{i \in \omega}$ such that $A = \bigcup_i B(c_i, r_i)$ where e_i is the code of the pair (c_i, r_i) . We say that $B \subseteq \mathbb{R}$ is *effectively closed* or Π_1^0 over \mathbb{R} if $\mathbb{R} \setminus B$ is effectively open.

Effectively G_δ set or Π_2^0 , *effectively F_σ set* or Σ_2^0 and Δ_2^0 over \mathbb{R} sets can be defined in a similar fashion as in Definition 2.18.

By Lemma 3.4, any TTE-computable function F is continuous, hence the pre-images $F^{-1}[\{x \in \mathbb{R} : x \neq 0_{\mathbb{R}}\}]$ and $F^{-1}[\{0_{\mathbb{R}}\}]$ are open and closed respectively. We verify that their indices can be effectively found⁷. Let $\varphi_e(i)$ denote the standard universal partial recursive function.

Lemma 3.12. *There is a recursive function $g : \mathbb{N} \rightarrow \mathbb{N}$ such that for any TTE-computable over \mathbb{R} function $\lambda x. \Phi^x(e)$, the set*

$$\{x \in \mathbb{R} : \Phi^x(e) \neq 0\} = \bigcup_{i \in \mathbb{N}} B(c_{\varphi_{g(e)}(i)}, r_{\varphi_{g(e)}(i)}),$$

hence is open.

Proof. Let M_e be the MS-machine that computes $\lambda x. \Phi^x(e)$. The following uniform effective procedure gives us the open set from the index e , whose code will be $g(e)$:

Step 1. For each $n \in \mathbb{Z}$, define a recursive tree R_n as follows: The root of R_n is n . Suppose that $\sigma \in R_n$ is defined and is of length s , then let $\sigma^\wedge(\sigma(s))$ and $\sigma^\wedge(\sigma(s) + \frac{1}{2^{s+1}})$ be the successors of σ on R_n . Clearly, each R_n satisfies the following properties: (1) For each $\sigma \in R_n$, σ is a finite nondecreasing dyadic Cauchy sequences with $\sigma(j) \in [n, n+1] \cap D_j$ for every $j < |\sigma|$; (2) for any $x \in [n, n+1]$ there is an infinite path δ of R_n such that $\lim \delta = x$. By simultaneously enumerate $(R_n : n \in \mathbb{Z})$, we are in fact enumerating the trees that contain all paths on T_α for all Cauchy $\alpha \in \mathbb{Q}^\omega$, where T_α is the tree produced by the fine-tuning procedure.

Step 2. For each $\sigma \in R_n$, let $s = |\sigma|$. Mimic the computation M_e^σ for s steps. If for some $j < s$, $M_e^\sigma(j) \downarrow [s]$ and is not C-compatible with 0^s , then enumerate the interval $(\sigma(s), \sigma(s) + \frac{1}{2^s})$ into O . (The first two steps take care of nondyadic x .)

Step 3. For each dyadic rational number d , let $\delta_L^d = (d - \frac{1}{k} : k \in \mathbb{N})$ and δ_R^d be the constant squence (d, d, \dots) . We can enumerate all finite initial segments of δ_L^d and δ_R^d for

⁷Similar results are done by computable analysts, see footnote 6.

all possible d . For each d , let $\delta_L := \delta_L^d$ and $\delta_R := \delta_R^d$. Mimic the computation $M_e^{\delta_L \uparrow s}$ and $M_e^{\delta_R \uparrow s}$ for s steps. If for some $j < s$, $M_e^{\delta_L \uparrow s}(j) \downarrow [s]$, $M_e^{\delta_R \uparrow s}(j) \downarrow [s]$ and both are Cauchy and their average is not C-compatible with 0^j , then enumerate the interval $(d - \frac{1}{2s}, d + \frac{1}{2s})$ into O . That finishes the algorithm.

We now verify that this algorithm works. If x gets enumerated, say in the second step via witness σ . Suppose $f(x) \downarrow$ and α is a Cauchy representative for x , then $M_e^\alpha \downarrow$. By the fine-tune procedure, σ is an initial segment of an infinite path on the tree T_α , thus $M_e^\alpha \neq 0$, because its first j bits are incompatible with 0^j . Similarly for those x enumerated in the third step.

On the other hand, suppose $f(x) \neq 0$. We consider the following two cases: Case 1. x is nondyadic. Then we know there is a unique dyadic sequence δ such that $\delta(i)$ is the biggest dyadic in D_i that is less than x . So M_e^δ must be incompatible with 0. By the stage that we discover the fact, x is enumerated into O . Case 2. x is a dyadic, say d . Then x is enumerated into O in step 3 by a similar argument. \square

Definition 3.13. We say that a subset $A \subseteq \mathbb{R}$ is *recursive over \mathbb{R}* if its characteristic function χ_A is partial recursive over \mathbb{R} .

As in Baire space, recursive over \mathbb{R} sets are closed under complementation.

Lemma 3.14. *Every effectively open subset A of \mathbb{R} is recursive over \mathbb{R} . So is every effectively closed set.*

Unlike in \mathcal{N} , where the membership of a basic open set $\llbracket \sigma \rrbracket$ can be decided by a single slave, in \mathbb{R} , it needs the collective effort of all slaves and a zero-test to determine the membership of $B(c, r)$ (see Claim 2 below). That is why the proof is much more complicated than the proof of Lemma 2.20, which is its counterpart over \mathcal{N} .

Proof. For a rational interval $I = (p, q)$, we say that L is a *linear function* on I if the domain of L is $[p, q]$ and $L(x) = \frac{s-r}{q-p}(x-p) + r$ for some rationals s and t , in other words, it is a line connecting (p, r) and (q, s) on the plane. We say that f is a *piecewise linear function*, if there are finitely many consecutive rational intervals $[p_0, p_1], \dots, [p_{n-2}, p_{n-1}]$ ($n \geq 2$) such that $f \upharpoonright [p_k, p_{k+1}]$ is a linear function L_k and $L_k(p_{k+1}) = L_{k+1}(p_{k+1})$.

Claim 1. Any piecewise linear function f is TTE computable.

Proof Sketch of Claim 1. Given any input α representing $x \in \mathbb{R}$, let $y_n = f(\alpha(n))$. Since the endpoints of the intervals are rational, it is recursive to decide $\alpha(n)$ belongs to which interval(s), so that we know which linear function to apply. And the continuity of f ensures that if $\alpha \sim \beta$ then $(f(\alpha(n)) \sim f(\beta(n)))$.

By Claim 1, we can show that

Claim 2. Any single rational interval $I = (p, q)$ is MS-computable.

Proof Sketch of Claim 2. Consider the following “zero-test preparing function” for I

$$f(x) = \begin{cases} 0, & \text{if } x \leq p \text{ or } x \geq q; \\ \frac{1}{q-p}(x-p), & \text{if } p \leq x \leq \frac{p+q}{2}. \\ -\frac{1}{q-p}(x - \frac{p+q}{2}) + \frac{1}{2}, & \text{if } \frac{p+q}{2} \leq x \leq q \end{cases}$$

which is TTE-computable by Claim 1. So we let the slaves to compute $f(x)$ and write the result on the zero-test tape. Since $x \in I$ if and only if $f(x) \neq 0$, by invoking the zero-test, we know if $x \in I$.

Note that the preparing function for a single open interval can be chosen to be the distance function d_C , where C is the closed set $\mathbb{R} \setminus I$. But for open sets in general, the distance function will not work. But luckily we do not require to find the distance if we just want to determine the membership of the open set.

Claim 3. Any effectively open subset U of \mathbb{R} is MS-computable.

Proof of Claim 3. Now let $U = \bigcup_n I_n$ be an effective open set and fix a recursive enumeration (I_n) . Without loss of generality, we assume that each I_n is bounded and we allow repetitions, so that we can assume that I_n is enumerated at stage n . Since we cannot afford to have infinitely many zero-tests, we need a preparing function f which works for the whole U . In other words, we need a function f satisfying (1) f is TTE computable; (2) $x \in U$ if and only if $f(x) \neq 0$; (3) the algorithm for computing $f(x)$ does not depend on the presentation α of x .

We will define a sequence (f_n) of piecewise linear functions and f will be the pointwise limit of f_n .

Let f_0 be the zero-test preparing function for I_0 as described in the proof of Claim 2. Suppose that f_n have been defined and $U_n = \{x : f_n(x) \neq 0\}$ which is a finite union of rational intervals. Look at I_{n+1} .

Case 1. $I_{n+1} \subseteq U_n$, then let $f_{n+1} = f_n$.

Case 2. $I_{n+1} \cap U_n = \emptyset$, then let

$$f_{n+1}(x) = \begin{cases} f_n, & \text{if } x \notin I_{n+1}; \\ \frac{1}{2^n} g_n(x), & \text{if } x \in I_{n+1} \end{cases}$$

where $g_n(x)$ is the zero-test preparing function for I_{n+1} defined as in the proof of Claim 2. The factor $\varepsilon = \frac{1}{2^n}$ is needed, because otherwise we may have some non-Cauchy sequence $y \in \mathbb{Q}^\omega$ with $y_s = 0$ and $y_{s+1} > \frac{1}{2^s}$ as the outcome.

Case 3. Neither Case 1 nor Case 2. Then define

$$f_{n+1}(x) = \begin{cases} f_n, & \text{if } x \notin I_{n+1}; \\ h_n(x), & \text{if } x \in I_{n+1} \end{cases}$$

where h_n can be viewed as a “ $\frac{1}{2^n}$ lifting of f_n within I_{n+1} ”. To be more precise, let f be a piecewise linear function, $I = (p, q)$ be a rational interval and $\varepsilon > 0$ is a rational number, we say that $h : [p, q] \rightarrow \mathbb{R}$ is an ε -lifting of f within I , if h is obtained from f as follows: Let $p \leq q_0 < \dots < q_k \leq q$ such that (q_i, q_{i+1}) are the domains of all linear pieces of f between p and q . Let $r = \max\{0, f(p)\}$, $s_i = f(q_{i+1}) + \varepsilon$ and $s = \max\{0, f(q)\}$. Then h is the piecewise linear function connecting $(p, r), (q_1, s_1), \dots, (q_k, s_k), (q, s)$ on \mathbb{R}^2 .

Since $|f_{n+1}(x) - f_n(x)| \leq \frac{1}{2^n}$ for all $x \in \mathbb{R}$, the sequence (f_n) converges to f uniformly. Consequently f is continuous. The definition above induces a TTE-procedure to compute f : Each slave S_i just compute $f_i(\alpha(i))$. Since the definition of f does not depend on α and f is continuous, this procedure is well-defined and Cauchy preserving. Finally, $x \in U$ iff $x \in U_i$ for some i iff $f(x) \neq 0$.

In summary, U can be computed using two master steps: Step 1: make the preparing function f and write $f(\alpha)$ on the zero-test tape; Step 2: execute zero-test and read off the answer. \square

Theorem 3.15. A subset A of \mathbb{R} is recursive over \mathbb{R} if and only if it is Δ_2^0 over \mathbb{R} .

Proof. “(\Leftarrow)” By the proof of Lemma 3.14, it takes only two master steps to determine if x is in a basic open set. So the whole proof of Lemma 2.21 can go through in the context of \mathbb{R} .

“(\Rightarrow)” By Lemma 3.12, we can uniformly find the indices of $\{x \in \mathbb{R} : \Phi^x(e) \neq 0\}$. So the whole proof of Theorem 2.24 can go through in the context of \mathbb{R} . \square

4. PROPERTIES OF MS-COMPUTABLE FUNCTIONS AND SETS

4.1. The decomposibility of MS-computable functions. Analyzing the proofs of Lemma 2.21 and Theorem 2.24, we see that the same analysis holds for any total MS-computable function (rather than the characteristic function of a set) over \mathcal{N} or \mathbb{R} . In fact, one can derive the following:

Corollary 4.1. *Let $F : \mathbb{R} \rightarrow \mathbb{R}$. Then F is (total) MS-computable if and only if there is an effective partition $\{X_i\}$ of \mathbb{R} into Δ_2^0 sets and an effective sequence of continuous functions $\{H_i\}$ such that for every i , $F \upharpoonright X_i = H_i \upharpoonright X_i$.*

Furthermore, each X_i can be taken to be the intersection of an effectively open set and an effectively closed set.

Our investigation is related to a well-known result in descriptive set theory, the Jayne-Rogers theorem. We can state the version for \mathbb{R} as the following:

Theorem 4.2 (Jayne, Rogers [7]). *Let $f : \mathbb{R} \rightarrow \mathbb{R}$. Then $f^{-1}(O)$ is Δ_2^0 for every open set O if and only if there is a partition of \mathbb{R} into Δ_2^0 sets $\{X_i\}$ such that $f \upharpoonright X_i$ is continuous for each i .*

Effective versions of the Jayne-Rogers theorem were also studied and discussed in Pauly, de Brecht [12]. Pauly and de Brecht showed that a certain effective version of the Jayne-Rogers theorem was true for computable metric spaces:

Theorem 4.3 (Pauly, de Brecht [12]). *A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is effectively Δ_2^0 -measurable if and only if it is piecewise computable.*

We note that the notion of being piecewise computable is the same as the existence of the decomposition in Corollary 4.1. Thus, we obtain:

Corollary 4.4. *A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is MS-computable if and only if it is effectively Δ_2^0 -measurable if and only if it is piecewise computable.*

4.2. Comparing TTE and MS-computability over \mathcal{N} . We now give some examples of nonrecursive sets over \mathcal{N} and \mathbb{R} .

Proposition 4.5. *The set $A = \{x \in \mathcal{N} : x \text{ has infinitely many zeros}\}$ is not MS-computable over \mathcal{N} .*

Proof. We reduce the Π_2^0 -complete set of natural numbers $\text{Inf} = \{e : W_e \text{ is infinite}\}$ to the set A .

Suppose that A is computed by the MS-machine M . We convert M to get another machine M_I to compute Inf as follows: For any input $e \in \mathbb{N}$, first ask each slave S_i to check if $W_{e,i}$ has more elements than $W_{e,i-1}$. If the answer is yes, then write a zero on the zero-test tape, otherwise write a one. Now apply M to get an answer, which will be the outcome of M_I . Clearly, W_e is infinite if and only if the zero-test tape has infinitely many zeros. So M_I correctly computes Inf , contradict Proposition 2.25. \square

Proposition 4.6. *The set \mathbb{Q} is not MS-computable over \mathbb{R} .*

Proof. Because \mathbb{Q} is not a G_δ set, the result follows from Theorem 3.15. \square

Proposition 4.7. *If $x \in \mathcal{N}$ and $M(x) \downarrow = y$, then there is a TTE-computable function f such that $f(x) = y$, i.e., there is a Turing functional $\lambda x.\Phi^x$ such that $\Phi^x \downarrow = y$. In particular, if x is a total recursive function over \mathbb{N} and $M(x) = y$, then y is a total recursive function over \mathbb{N} . However the map from the index of x to the index of y is \emptyset' -recursive.*

Proof. (sketch) During the computation of $M(x) = y$, M only uses finitely many zero-test steps. We may code the answers of the zero test results as a single parameter, then the computation becomes TTE. \square

However, there are some subtleties.

Proposition 4.8. (a) *The singleton set $\{\chi_{\emptyset'}\}$ is not recursive over \mathcal{N} (as a subset of \mathcal{N}), even though \emptyset' is recursive over \mathcal{N} (as a subset of ω).*
 (b) *Let $f \in \mathcal{N}$ be such that $f(n) = 0$ if $n \notin \emptyset'$, and $f(n) = s$ if $n \in \emptyset'$ and s is the least stage of n entering \emptyset' . Then $\{f\}$ is recursive over \mathcal{N} . (This $\{f\}$ is a Π_1^0 singleton coding \emptyset' in \mathcal{N} .)*

Proof. (a) Suppose that $\{\chi_{\emptyset'}\}$ is recursive over \mathcal{N} . Then by Theorem 2.24, $\{\chi_{\emptyset'}\}$ is a Σ_2^0 -singleton, hence it is a Π_1^0 -singleton (by taking the Σ_2^0 -witness as a parameter) in the Baire space. In other words, there is a recursive tree S , with $\chi_{\emptyset'}$ as its unique infinite branch. However, $\chi_{\emptyset'}$ is 0-1 valued, so by restricting S to the Cantor space, we get a recursive binary tree T with $\chi_{\emptyset'}$ as its unique infinite branch, which is obviously impossible.

(b) Let M be the following MS-machine: For any input x , let the i -th slave check the correctness of the first i bits of x up to step i . To be more precise, for each $j < i$, if $x(j) = 0$ then check whether $j \notin W_{j,i}$; if $x(j) = s \neq 0$, then check if s is the least stage that $j \in W_{e,s}$. If the i -slave found an error, then writes a one on the zero-test tape, otherwise, writes a zero. Clearly $x = f$ if and only if the sequence written on the zero-test tape is the zero sequence. By applying zero-test, the master knows if $x = f$. \square

Proposition 4.8 tells us: (1) A set $A \subseteq \omega$ being a recursive set is different from its characteristic function χ_A being a MS-computable singleton in \mathcal{N} . (2) It is possible to have two elements x and y in the Baire space, which are TTE-equivalent, i.e., there are Turing functionals Φ and Ψ with $\Phi^x = y$ and $\Psi^y = x$, but $\{x\}$ is MS-computable, whereas $\{y\}$ is not.

4.3. Comparing MS-computability over \mathcal{N} and over \mathbb{R} . Despite the computable sets in \mathcal{N} and in \mathbb{R} being Δ_2^0 -definable within the structure, their differences are striking. One of the reasons is that \mathcal{N} is not locally compact whereas \mathbb{R} is.

The following folklore is well-known:

Fact 4.9. For each recursive ordinal α , there is some $f \in \mathcal{N}$ such that $f \equiv_T \emptyset^{(\alpha)}$ and f is a Π_1^0 -singleton.

Proof. By Sacks [13, Theorem II.4.2], for each computable ordinal α , $\emptyset^{(\alpha)}$ is a Π_2^0 -singleton. By Jockusch and McLaughlin, [8, Theorem 3.1], we can replace each Π_2^0 -singleton with a Turing equivalent Π_1^0 -singleton. \square

We compare the singletons that are MS-computable in \mathcal{N} and in \mathbb{R} . Each MS-computable singleton in \mathbb{R} must be a recursive real, and by Fact 4.9, the MS-computable singletons in \mathcal{N} are far from being recursive:

- Lemma 4.10.** (i) $\{f\} \subseteq \mathcal{N}$ is MS-computable if and only if f is a Π_1^0 -singleton.
(ii) $\{x\} \subseteq \mathbb{R}$ is MS-computable if and only if x is a recursive real.

Proof. (i): Each Π_1^0 -class in \mathcal{N} is MS-computable by Lemma 2.21. On the other hand, if $\{f\}$ is MS-computable then it is a Δ_2^0 class by Theorem 2.24, and each Σ_2^0 -singleton is also Π_1^0 .

(ii): The right-to-left direction is obvious. Now suppose that $\{x\}$ is MS-computable, hence it is effectively Δ_2^0 , and hence effectively closed. However, each effectively closed singleton in \mathbb{R} is clearly both left-r.e. and right-r.e. □

A set $X \subseteq Y$ is said to *determine* a class of functions with domain Y , if for any two functions F and G from this class, $F \upharpoonright X = G \upharpoonright X$ implies that $F = G$. For instance, the class of TTE-computable functions is determined by the class of dyadic rationals.

Next, we compare MS-computability on \mathcal{N} and on \mathbb{R} by comparing the least complicated set that determines MS-computability. Again, we see that MS-computable functions on \mathbb{R} can be represented much more simply than MS-computable functions on \mathcal{N} :

- Proposition 4.11.** (i) The class of MS-computable functions on \mathcal{N} is determined by the class of all $f \in \mathcal{N}$ with $f \leq_T \mathcal{O}$ (the Kleene's \mathcal{O}). Furthermore, for any recursive ordinal α , the class Δ_α^0 does not determine the class of MS-computable functions on \mathcal{N} .
(ii) The class of MS-computable functions on \mathbb{R} is determined by the class of all Δ_2^0 reals (in fact, low reals), but not by the class of all computable reals.

Proof. (i): If F and G are MS-computable, then $F(\alpha) \leq_T \alpha'$, and thus $F(\alpha) \neq G(\alpha)$ is an arithmetical predicate. By Kleene's Basis Theorem (see [13, Theorem 1.3 on Chapter 3]), this is witnessed by some $\alpha \leq_T \mathcal{O}$. On the other hand, by Fact 4.9 and Lemma 4.10, for any recursive ordinal α , there are MS-machines M_d and M_e computing the sets $\{F\}$ and $\{G\}$ for some $F \equiv_T \mathcal{O}^{(\alpha+1)}$ and some $G \equiv_T \mathcal{O}^{(\alpha+2)}$. So they differ only at the points F and G and in particular, they are equal on all inputs recursive in $\mathcal{O}^{(\alpha)}$.

(ii): By Corollary 4.1, if F and \hat{F} are MS-computable and $F(x) \neq \hat{F}(x)$ for some real x , then for some Δ_2^0 sets X, \hat{X} , and some effectively continuous H and \hat{H} , we have $x \in X \cap \hat{X}$, $F(x) = H(x)$ and $\hat{F}(x) = \hat{H}(x)$. Since $H(x) \neq \hat{H}(x)$, let I be a small enough rational interval containing x such that $(H \upharpoonright I) \cap (\hat{H} \upharpoonright I) = \emptyset$. Now consider $X \cap \hat{X} \cap I$; this is a non-empty Δ_2^0 -class in \mathbb{R} , and therefore must contain some low member, say \hat{x} . But this means that $F(\hat{x}) = H(\hat{x}) \neq \hat{H}(\hat{x}) = \hat{F}(\hat{x})$. On the other hand, there is an effectively closed set C which has no recursive member, any MS-machine which computes C will agree with constant zero function on all recursive real. □

5. CONCLUDING REMARKS

There is an interesting debate about whether algorithms should be defined in terms of abstract machines (Gurevich [5] [6]) or in terms of "recursor" (Moschovakis [11]). Vardi made some inspiring remarks in his short article [17]. He mentioned the de Broglie's *wave particle duality* in physics and claimed

An algorithm is both an abstract state machine and a recursor, and neither view by itself fully describes what an algorithm is. This algorithmic duality seems to be a fundamental principle of computer science.

This duality occurred in the classical definition of algorithms over natural numbers, and it occurred again in our analysis of computability over real numbers. Our analysis seemed to suggest that the classical correspondence between the recursion schemes and Turing machines can be applied to other domains. Both the recursion scheme and the Turing machine can be viewed as control units above some domain-dependent primitive functions or operations.

However, our work heavily depends on the underlying algebraic and topological structure of real numbers, and the machine part also relies on representing a real number as an ω -sequence. Comparing to the natural and elegant formalizations of computability over natural numbers, in particular the works by Gödel [4] and Turing [15], more insights are needed. The ultimate question remains to be: Is there a natural, elegant and general definition of algorithms over all domains? If so, what is it?

REFERENCES

- [1] Blum, L., Shub, M. and Smale, S. On a theory of computation over the real numbers: NP completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc.* (N.S.) **21**, no. 1, 1–46, 1989.
- [2] Brattka, V., The emperor’s new recursiveness: the epigraph of the exponential function in two models of computability, in *Words, languages & combinatorics, III (Kyoto, 2000)*, 63–72, World Sci. Publ., River Edge, NJ, 2003.
- [3] Gödel, Kurt. Über formal unentscheidbare Sätze der *Principia mathematica* und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, **38**, 173–198, 1931.
- [4] Gödel, Kurt. *On undecidable propositions of formal mathematical systems* (mimeographed lecture notes, taken by Stephen C. Kleene and J. Barkley Rosser) 1934; reprinted with revisions in *The undecidable: basic papers on undecidable propositions, unsolvable problems, and computable functions* M. Davis (ed.) 39 – 74, Raven Press, NY, 1965.
- [5] Gurevich, Yuri. Sequential Abstract State Machines capture Sequential Algorithms. *ACM Transactions on Computational Logic* Volume 1, Number 1, 77–111, 2000.
- [6] Gurevich, Yuri. “What is an algorithm?” in *SOFSEM: Theory and Practice of Computer Science* (eds. M. Bielikova et al.), Springer LNCS **7147**, 31–42, 2012.
- [7] Jayne, J. E., and Rogers, C. A. First level Borel functions and isomorphisms. *J. Math. Pures Appl.*, (9) **61** no. 2, 177–205, 1982.
- [8] Jockusch, C. G., Jr. and McLaughlin, T. G. Countable retracing functions and Π_2^0 predicates *Pacific J. Math.*, **30**, 67–93, 1969.
- [9] Kleene, Stephen C. General recursive functions of natural numbers. *Mathematische Annalen* **112**, 727 – 742, 1936.
- [10] Moschovakis, Yiannis N. Abstract first order computability I, *Trans. Amer. Math. Soc.* **138**, 427–464, 1969.
- [11] Moschovakis, Yiannis N. “What is an algorithm?”, in *Mathematics Unlimited—2001 and beyond* (eds. B. Engquist and W. Schmid), Springer, 919–936, 2001.
- [12] Pauly, Arno, and de Brecht, Matthew. Non-deterministic computation and the Jayne-Rogers Theorem. *Electronic Proceedings in Theoretical Computer Science*. **143**, 87 – 96..
- [13] Gerald E. Sacks. *Higher Recursion Theory*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1990.
- [14] Robert I. Soare. *Recursively Enumerable Sets and Degrees*. Springer-Verlag, Heidelberg, 1987.
- [15] Turing, A. M. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proc. London Math. Soc.*, S2-42 (1): 230–265, 1936.
- [16] Turing, A.M. Systems of logic based on ordinals, *Proc. London Math. Soc.*, 45 Part 3: 161—228, 1939.
- [17] Vardi, Moshe Y. What is an algorithm? *Communications of the ACM*, **55** No. 3: p.5, 2012.

- [18] Weihrauch, Klaus. *Computable Analysis, an introduction*. Springer-Verlag, Berlin, 2000.
- [19] Yang, Y. A Turing machine like model for computation on real numbers. To appear in the Lecture Notes Series. Institute for Mathematical Sciences. National University of Singapore. World Scientific Publishing.