Introducing Computability of Topological Spaces, with Applications to $\mathbb{R}^n$

Boxian Wang

November 18, 2022

1 Introduction

1.1 Why Do We Care?

The notion of computability arises from the study of functions that can be 'effectively' calculated. The concept of 'effective' or 'algorithmic' computability has been popularized via the wide adoption of electronic computers, which are capable of completing billions of programmed calculations in fractions of a second, whereas in Turing's days one had to resort to a tireless yet precise scribe to meticulously carry out the instructions.

Computability theory not only serves as the bedrock on which modern computer science is built, but also provides insights into other areas of mathematics. Specifically, for any abstract mathematical object, one can attempt to ask the question whether it is 'computable' in some way, with a suitable definition of computability.

The 'vanilla' computability theory is based on $\mathbb{N}$ and functions $\mathbb{N} \to \mathbb{N}$. This is natural as computers deal with finite objects only. $\mathbb{N}$ may seem tame at first glance, but it is rather a surprising result that some subsets (and functions) over $\mathbb{N}$ can be so malformed that they are uncomputable. Hence, when generalizing to more abstract, potentially infinite objects, one can anticipate a fruitful investigation into the computability of those objects and their mappings. For example, what would be a suitable concept of computability over $\mathbb{R}$? What consequences does that notion of computability have on the topological structure of Euclidean spaces? How do we generalize to metric spaces or topological spaces in general?

To give a concrete motive, we only need consider some of the more extreme pathologies in general topology. Take the Mandelbrot set[4] for example. What does its complex and fractal nature say about its computability?

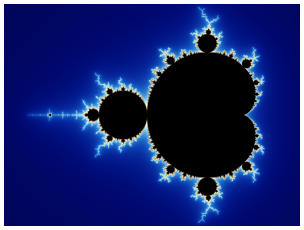


Figure 1: The Mandelbrot set[1]

1.2 Outline

In this report, I shall provide some initial insights into these problems, which hopefully leads to further applications and theoretical studies. I shall construct the theory of computability in topological spaces in 4 parts. Part 2 gives an overview of the concepts and crucial results of computability theory. Part 3 extends the basic computability theory to abstract objects, with examples in $\mathbb{R}$. Part 4 describes the theory of computability, in increasing level of generality, of Euclidean spaces, metric spaces and topological spaces. Part 5 points to advanced applications, future studies and potential research topics.

2 Basic Computability

2.1 Models of Computation

There are many ways of defining what 'effective' or 'algorithmic' computation is, the classic being Turing Machine. However, I prefer the RAM model[8] as it more closely resembles the actual computer.

Definition 2.1. *A RAM machine R consists of an infinitely long work tape (indexed by natural numbers) capable of holding integers and finitely many instructions, $D_1, \ldots, D_k$.*

There are four types of instructions:

1. *Zero. $Z(n)$ sets position n to 0.*
2. *Increment. $I(n)$ adds 1 to position n.*
3. *Swap. $S(n, m)$ swaps contents in n, m.*
4. *Jump. $J(n, m, p)$ goes to instruction D_p if contents in n, m are equal. Otherwise, proceed.*

*The computation starts with an initial work tape $a_1, \ldots, a_t, 0, 0, \ldots$ with finitely many non-zero inputs. The RAM machine modifies the work tape per the instruction, and go to the next instruction sequentially, unless a jump instruction is encountered, in which case it jumps to the specified instruction conditionally. We say a computation **halts** when the machine moves beyond the last instruction D_k.*

Note that a computation may or may not halt. If it halts, the result of the computation is defined to be the content in position 1.

With a machine model that reasonably resembles a real computer, we proceed to define computable functions on $\mathbb{N}^n$. It is a peculiarity of computability theory that we don't require a function to be defined on all of $\mathbb{N}^n$;it can be undefined at some places. We shall see why this is important.

Definition 2.2. *A function $f : \mathbb{N}^n \to \mathbb{N}$ is called **computable** if there exists a RAM machine R such that:*

For every $(a_1, \ldots, a_n)$ where f is defined, with mapped value v , R halts on the input $a_1, \ldots, a_n, 0, \ldots$ with result v.

For every input where f is undefined, R does not halt.

Perhaps it is better to call such a function RAM-computable, because at first glance our computing model seems rather arbitrary. However, our definition turns out to possess significant generality. In fact, any 'reasonable' formulation of computability does. This is the famous *Church-Turing thesis*.

Theorem 2.3. *(Church-Turing thesis) All intuitive notions of effective computability are equivalent to the RAM model (or whatever your favourite model is).*

There is, of course, no formal proof to such sweeping generalization. However, it is often used as the belief that any intuitively computable function *can somehow* be modeled by a RAM machine. Of course, mathematicians don't bother to provide a rigorous proof each time, because that would be the job of software engineers who get paid top dollars to do just that. Instead, they just appeal to CT-thesis.

Another interesting thing is that we can create a universal machine capable of emulating any machine. But to do that, we shall first see that we can 'encode' RAM-machines. Not in C or Python or assembly, but in natural number.

Theorem 2.4. *There is a bijection G from $\mathbb{N}$ to the set of all RAM-machines. That is, RAM-machines are countably infinite. G is often called a* ***Gödel numbering****.*

The proof seems quite clear, as a RAM machine is just a finite sequence of instructions, which are themselves countable. Deriving a concrete numbering is tedious but possible, and we omit it here and just assume we have such a numbering.

By the definition of computable function, we also get a numbering of computable function, indexed $\phi_1, \phi_2, \ldots$ Note that the numbering is not injective.

Theorem 2.5. *There exists a two variable computable function ϕ, called the* ***universal machine****, such that $\phi(x, y) = \phi_y(x)$*

To paraphrase, ϕ takes an input and the encoding of a program, and outputs the result of running that program on the input. But this is exactly what a computer does! Hence we appeal to CT and claim to have proved the theorem.

2.2 Incomputability; Recursive and R.E. Sets

We now give an example of an uncomputable function, which relies on the fact that computable functions are countable and we may use Cantor's diagonal argument.

Theorem 2.6. *Let*

$$\phi(x) = \begin{cases} 1, & \textit{if } x \textit{ is in the domain of } \phi_x \\ 0, & \textit{otherwise} \end{cases}$$

Then ϕ is uncomputable.

Proof. If ϕ is computable, we create a new computable function.

$$\psi(x) = \begin{cases} 0, & \phi(x) = 0 \\ undefined, & \text{otherwise} \end{cases}$$

This leads to a contradiction, as ψ differ with each ϕ_x at x, whence it is not computable. □

This is expected as $\mathbb{N}^{\mathbb{N}}$ is uncountable, which implies there are uncountable numbers of uncomputable functions. We have the famous halting problem as a corollary:

Corollary 2.7. *The function*

$$\phi(x,y) = \begin{cases} 1, & x \text{ in the domain of } \phi_y \\ 0, & otherwise \end{cases}$$

is uncomputable.

That is to say, it is impossible to algorithmically determine whether a given program y halts on a given input x. To find a prove, simply notice that the computability of the halting problem particularly implies that the function in the previous theorem is computable.

Finally, we may reframe the computability of functions as a property on subsets of $\mathbb{N}$.

Consider the total characteristic function of a subset $S \subset \mathbb{N}$:

$$c_S(x) = \begin{cases} 1, & x \in S \\ 0, & x \notin S \end{cases}$$

Definition 2.8. *S is **recursive** if c_S is computable.*

We may weaken the condition somewhat by defining a weak characteristic function d_S:

$$d_S(x) = \begin{cases} 1, & x \in S \\ undefined, & x \notin S \end{cases}$$

Definition 2.9. *S is **recursively enumrable**, or **r.e.**, if d_S is computable.*

Finally, we give a theorem[8] that characterizes r.e. sets.

Theorem 2.10. *The following are equivalent:*

1. *A is r.e.*
2. *A is the domain of a computable function.*
3. *$A = \emptyset$ or A is the range of a total computable function.*

Furthermore, a set X is recursive if and only if both X and $\mathbb{N} - X$ are r.e.

We shall omit the proof but point out that the theorem states A can be obtained by enumerating values of a recursive (computable) function, hence the name recursively enumerable.

3 Extending Computability

3.1 Some Background

Now that we have a notion of computability on $\mathbb{N}$, we shall attempt to generalize that notion to other mathematical object, with a possibly infinite nature. A good target is $\mathbb{R}$, which interestingly was the consideration in Turing's original paper[13], which adopted the decimal representations of real numbers. Despite the greatness of that paper, his approach was not entirely satisfactory. However, unlike the canonical computability theory, which has equivalent, commonly accepted foundations, computability in $\mathbb{R}$ and in general has had multiple non-equivalent formulations. We shall attempt to follow the development of Weihrauch[16], and start with more general formulations before zeroing in on $\mathbb{R}$ as a special case.

3.2 Computability of Infinite Sequences

To study computability on arbitrary abstract object, we consider representing it with a sequence of finite objects. Specifically, let Σ be a finite alphabet, and let Σ^ω be the set of sequences of symbols in the alphabet, and let Σ^* be the subset of elements with finite support (i.e. finite length words). Note that we have already defined computability on Σ^* due to its countable nature.

We would like to define computability on infinite sequences. There are many non-equivalent ways to proceed here, but we shall define a notion of computability[19] no stronger than the usual version. Specifically, we impose the seemingly severe restriction that, for a computation with infinite input and output, any **finite** portion of the output must be determined by a **finite** portion of the input.

This restriction on the surface seems so severe that multiplication of a decimal number by 3 turns out to be uncomputable. Intuitively, the infinite decimal $0.\dot{3}$ time 3 equals the infinite decimal $0.\dot{9} = 1.0$, but we cannot decide what the digit before the decimal point should be by just looking at a finite portion of the input, as any digit $\neq 3$ after that portion may alter the result!

Quite surprisingly, that defect does not undermine the validity of our definition of computability. Rather, it can be shown the problem lies in the decimal representation of $\mathbb{R}$.

Thus, we proceed with a simplified definition of the aforementioned computability. Let $w < v$ denote "w is a prefix of v".

Definition 3.1. *A function $f : \Sigma^\omega \to \Sigma^\omega$ is called computable if there exists a computable function $g : \Sigma^* \to \Sigma^*$, such that for $w < v$, $g(w) < g(v)$, and that g 'approximates' f.*

That is to say, for every sequence p where f is defined, any finite prefix of $f(p)$ can be obtained by applying g to a finite prefix of p.

Furthermore, a modified Turing machine, called *Type-II Turing machine*[16], doing infinite computations on a one-directional output tape could be viewed as equivalent to the notion of computability defined above.

This leads to the notion of 'computable sequences', or in case of $\mathbb{R}$, 'computable numbers'.

Definition 3.2. *A sequence $p \in \Sigma^\omega$ is **computable** if it is the output of a constant computable function.*

The usual closure properties of computable function (composition, recursion), still holds, and we may also generalize to multiple variables. Similarly, versions of universal machine theorem and s-m-n theorem hold as well. We omit the details here.

The last step is defining analogies of recursive and r.e. sets.

Definition 3.3. *A subset of Σ^ω is r.e. if it is the domain of a computable function. It is recursive if both itself and its complement are r.e.*

It turns out our definition of computability is **"topologically nice"**. We shall see why as we progress, but for now we state an important topological implication.

Consider the Cantor topology on Σ^ω, defined as the product of *discrete* topology on Σ. (The discrete topology on a set is where all subsets are open.) From now on we tacitly admit that continuity in Σ^ω is defined via Cantor topology.

Theorem 3.4. *Given the Cantor topology on Σ^ω, every computable function $f : \Sigma^\omega \to \Sigma^\omega$ is continuous.*

Proof. (Sketch) We point out that $\{w\Sigma^\omega, w \text{ finite}\}$ is a basis of the Cantor topology. These are sets of sequences with a finite fixed portion. Consider now $f^{-1}(w\Sigma^\omega)$. By our finiteness property, a finite portion of output is only determined by a finite portion of input. Thus $f^{-1}(w\Sigma^\omega) = v\Sigma^\omega$, which is open. □

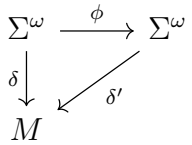
We are glad to see that topology gives us a surprisingly neat characterization of our supposedly ad-hoc notion of computability!

3.3 Sequence Representation

We now translate computability of sequences to computability of abstract objects via a representation, that is, a mapping between sequences and abstract objects. This correspondence is also aptly dubbed a 'naming system'.

Definition 3.5. *A representation of set M is a (partial) surjection $\delta : \Sigma^\omega \to M$.*

Some times one representation is related to another. It is best shown with a commutative diagram:



Definition 3.6. *Say $\delta < \delta'$, or δ reducible to δ', if ϕ is computable in the commutative diagram. Define "$<_t$", "continuously reducible" analogously if ϕ is continuous. If $\delta < \delta'$ and $\delta' < \delta$, say they are equivalent, or $\delta \equiv \delta'$. Continuous equivalence is similarly defined.*

Intuitively, ϕ serves as the translation between "naming systems", which can be imagined as natural languages (albeit infinite). If δ can always be translated to δ', δ' is *richer* than δ. Clearly "$<$" is transitive and $\equiv$ is an equivalence relation. The equivalence classes can be thought of as the classes of all mutually untranslatable languages. Lost in translation is real!

Now the sequence representation allows us to discuss computability on arbitrary M. The following definitions are natural.

Definition 3.7. *Let δ be a representation $\Sigma^\omega \to M$.*

1. *An element $x \in M$ is called δ-computable if it is represented by a computable element y: $x = \delta(y)$.*

2. *For a function $f : M \to M$, a function $g : \Sigma^\omega \to \Sigma^\omega$ is called its realization if the following diagram commutes:*

$$\begin{array}{ccc} \Sigma^\omega & \xrightarrow{g} & \Sigma^\omega \\ {\scriptstyle\delta}\downarrow & & \downarrow{\scriptstyle\delta\ (\delta')} \\ M & \xrightarrow{f} & M \end{array}$$

   f is called δ-computable (continuous) if g is computable (continuous). This generalizes to the case where M is represented by two different maps δ, δ', in which case (δ, δ')-computability is analogously defined.

3. *A set $X \subset M$ is called δ-recursive (r.e., open) if $\delta^{-1}(X)$ is recursive (r.e., open)*

Note that the above concepts generalize easily to multi-variable cases (with multiple representations)

We appeal to intuition rather than a formal proof to state that two representations are equivalent if and only if they induce the same notion of computability.

Here are some examples for a quick sanity check:

Example 3.8

1. *Let ρ_{dec} be the decimal representation of $\mathbb{R}$. Then $\pi = 3.14159\ldots$ is computable. So is $e = 2.71828\ldots$. Division by 3 is computable, but mulplication by 3 is not!*

2. *Let v_Q be the usual fractional representation of $\mathbb{Q}$. The sets $\mathbb{Z}, \mathbb{Q}_+$ are v_Q-recursive, and so is the set $\{x \in \mathbb{Q}, x^2 < 2\}$. Addition, subtraction, multiplication and division are all computable.*

3. *Consider two representations of $2^{\mathbb{N}}$ (subsets of $\mathbb{N}$): c, the usual characteristic function for subsets, and d, the weird one. d looks at a sequence of 0 and 1, and say n is in the subset if and only if the sequence contains section of the form $10^{n+1}1$. It can be shown that $c < d$ but not the other way around. Set complement is c-computable but not d-computable.*

3.4 Representations of $\mathbb{R}$

The previous example may be fun, but let us look at something more substantial. We already know that the decimal representation is not good enough, as it does not admit even a computable multiplication. Indeed, its definition is "ad-hoc", relying only on human anatomy rather than any intrinsic mathematical properties. Another less trivial (more fun) but still ad-hoc one would be the continued fraction representation[2], e.g

$$e = 2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \dots}}}$$

If we were to determine a more "natural" representation , we would have to rely on certain topological properties of $\mathbb{R}$. More on that in the next section, and for now we only describe one such representation and show that our model is not entirely hopeless when dealing with $\mathbb{R}$.

Definition 3.9. *The Cauchy representation $\rho_C : \Sigma^\omega \to \mathbb{R}$ is defined in the following way:*

Let $p \in \Sigma^\omega$ be viewed as a sequence of (representation of) rational numbers w_i. $\rho_C(p) = x$ if and only if $|w_i - w_k| < 2^{-i}$ for $k > i$ and $x = \lim w_i$.

Essentially we express $x \in \mathbb{R}$ as a Cauchy sequence of rational numbers with *known rate of convergence*. Knowing how fast the sequence converges is important, as intuitively it allows us to obtain any precision of output using a fixed portion of input.

We could go into details of representations of $\mathbb{R}$, but it is better to defer to later sections after seeing its interaction with topology in $\mathbb{R}$. For now we shall be content with showing multiplication is computable under Cauchy representation.

Theorem 3.10. *$(x, y) \to xy$ is computable under Cauchy representation.*

Proof. Consider the Cauchy representation of x, y: $\{x_i\}$, $\{y_i\}$. Let m be the smallest natural number such that $|x_0|, |y_0| \leq 2^{m-1}$. Then we have $|x_i| \leq |x_i - x_0| + |x_0| \leq 2^{m-1}$.

We now define a machine that works on $\{x_i\}$, $\{y_i\}$. It writes $s_i = x_{i+m}y_{i+m}$. Note this satisfies the finiteness condition as m is finite in all cases. Now naturally $\lim s_i = xy$ by basic calculus. We only need show the extra condition holds. For $k > i$,

$$\begin{aligned} |s_i - s_k| = |x_{i+m}y_{i+m} - x_{k+m}y_{k+m}| &\leq |x_{i+m}(y_{i+m} - y_{k+m})| + |y_{k+m}(x_{i+m} - x_{k+m})| \\ &< 2 \cdot 2^{m-1} \cdot 2^{-m-i} = 2^{-i} \end{aligned}$$

as desired. □

We now list, but not prove, some facts about computability under Cauchy representation[15]. These results should hopefully be compatible with the intuitive notion of real number 'computability'.

Example 3.11

I. Computable numbers in $\mathbb{R}$

(1) rational numbers
(2) $\sqrt{2}$
(3) $\log_2 10$
(4) e
(5) π

II. Computable functions in $\mathbb{R}$

(1) constant maps
(2) projection maps
(3) $-x$
(4) $(x, y) \to x + y$
(5) $(x, y) \to xy$
(6) $1/x$
(7) $\exp x$
(8) $\log x$
(9) $\sqrt{x}$
(10) $\sin x$

Note that from now on when we talk about computable elements, recursive/r.e. sets or computable function in $\mathbb{R}$, we mean those under ρ_C.

For those familiar with analysis, a natural next step is to consider limits of computable numbers (functions), as well as power series. It is a pleasant result that these behave exactly like they should, although with some extra computability constraints attached. To avert the danger of venturing too far into analysis, I shall stop here.

To steer our way back to topology, I shall admit that I did not provide adequate justification that the Cauchy representation is 'the' representation of $\mathbb{R}$. In fact, the representation $\rho_{C'}$ which does not require the convergence rate constraint admits a similarly nice array of computable numbers and function. However, it can be shown that $\rho_{C'}$ is not equivalent to ρ_C. To see the real reason, one would have to consider the interplay between topological spaces and computability.

4 Computability of Topological Spaces

4.1 Admissible Representation

We have seen that not all representations are useful: certain ad-hoc representations such as ρ_{dec} are of very little use. Ideally we would want a representation to be compatible with a certain (nice) topology.

Definition 4.1. *Let* (M, τ) *be a* T_0*-space with a countable subbasis* σ*. Let* $\rho : \Sigma^* \to \sigma$ *be a (finite) notation for* σ*. Call the tuple* (M, τ, ρ) *a* ***computable topological space*** *if the equivalence problem* $\{(u, v) | \rho(u) = \rho(v)\}$ *is r.e.*

T_0-space just means two points x, y can be told a part by looking at open sets they are in.

Note that the countability criterion stipulates that M is a second countable T_0-space. Essentially this guarantees $x \in M$ is uniquely decided by a family of subsets $S \in \sigma$ such that $x \in S$.

We turn that idea into a concrete representation of M.

Definition 4.2. *Use the same notations as in definition 4.1.*
Let $\delta_S : \Sigma^\omega \to M$ be defined as:

$$\delta_S(p) = x \iff \{A \in \sigma \mid x \in A\} = \{\rho(w) | w \text{ a subword of } p\}$$

δ_s is called the standard representation of M.

To interpret this definition, note that we are simply saying " p denotes x if and only if p includes all names for $A \in \sigma$ that contains x".

To justify the word 'standard', we give some very nice properties[16] of δ_S to offer some intuition.

Theorem 4.3. *Let's remind ourselves that the Cantor topology τ_C is assumed to be the standard one on Σ^ω. Assume (M, τ, ρ) has standard representation δ_S.*

1. *δ_S is (τ_C, τ)-continuous.*
2. *δ_S is an open map.*
3. *τ is the final topology of δ_S (finest topology such that δ_S continuous).*
4. *$\zeta <_t \delta_S$ for all continuous $\zeta : \Sigma^\omega \to M$.*

Since the standard representation is so nice, we have the following definition:

Definition 4.4. *A representation γ is called admissible with respect to (M, τ) if $\gamma \equiv_t \delta_S$, i.e. it is continuously reducible to the standard representation.*

And we summarize the nice properties of admissible representation in the following theorem:

Theorem 4.5. *Let δ be a representation of (M, τ).*
δ is admissible w.r.t. $\tau \iff \delta$ is continuous and $\zeta <_t \delta$ for all continuous representation ζ

Enough about admissible representation *per se*. What about the implications of the computability induced by such nice representation? The following theorem shows that our definition of admissible representation correctly bridges the topology on M and that on Σ^ω.

Theorem 4.6. *Let δ_1, δ_2 be admissible representations of $(M_1, \tau_1), (M_2, \tau_2)$.*
Let $f : M_1 \to M_2$. Then:
f is continuous (as defined by τ_1, τ_2) $\iff$ f is (δ_1, δ_2)-continuous.

Thus, under admissible representations, a continuous function can always be realized by a continuous function on the sequence space.

Remember our definition of computability implies continuity. This corroborates the topological 'nicety' of our definition of computability in our punchline:

Corollary 4.7. *Under admissible representations, computable functions are continuous (in the usual sense).*

4.2 $\mathbb{R}$ Revisited

Now is the time to reveal why ρ_C is the natural representation of $\mathbb{R}$. You probably have guessed it already!

Let $\tau_{\mathbb{R}}$ be the usual topology on the real line defined either with metric $|x-y|$ or with basis $\{(a,b)\}$ comprised of open intervals. Note that the concept from the last section may apply as $(\mathbb{R}, \tau_{\mathbb{R}})$ has a countable bases, namely the open intervals with rational endpoints.

Theorem 4.8. *ρ_C is admissible with respect to $(\mathbb{R}, \tau_{\mathbb{R}})$.*

Proof. Let us remind ourselves what the standard representation of $(\mathbb{R}, \tau_{\mathbb{R}})$ is: it locates a point x by inputting **all** rational open intervals that contains x. We claim that this is equivalent to locating a point by inputting nested (rational) intervals.

We need to find a way to translate one input to another that respects the finiteness constraint. One direction is simple: to find a nested interval representation given an enumeration of all open intervals containing x, simply take the first one, search until a nested one is found, and repeat.

The other direction is not too hard either. Given a set of nested interval, simply search for all open intervals containing the first nested interval. Upon failure, move to the second nested interval, and repeat.

Thus we shall prove the Cauchy representation in equivalent to the nested interval representation. We similarly show the existence of a two-way translation.

Given a controlled Cauchy sequence $\{w_i\}$, simply write down intervals $(w_i - 2^i, w_i + 2^i)$ in sequential order. Given a nested interval, to find w_i, simply search for an interval with length less than 2^{-i}, so that it guarantees any future find is at most 2^{-i} away. Repeat. □

This fully demonstrates the naturality of ρ_C as the $(\mathbb{R}, \tau_{\mathbb{R}})$ is the most natural topology on $\mathbb{R}$.

The naturality is further enhanced by the corollary:

Corollary 4.9. *Under Cauchy representation, every computable function is continuous.*

However, those more familiar with general topology may recall there are more esoteric topologies on $\mathbb{R}$ besides the usual one. Of special interest in computability theory are the topologies generated by $\{(x, +\infty)\}$ and $\{(-\infty, x)\}$ respectively. The corresponding standard representations are denoted $\rho_<$, $\rho_>$ resp.

It is fruitful to investigate computability concept induced by these representations, and it can be shown that ρ_C is the conjunction[16] of $\rho_<$ and $\rho_>$.

4.3 Computability of Open and Compact Sets

Finally, we shall take a look at representing open and compact subsets of $\mathbb{R}$[4, 15]. This will enable us to investigate interesting problems outlined in the introduction. We will begin with open and closed sets.

Definition 4.10. *Define δ_o the standard representation of open sets as follows. Let p encode rational open intervals $\{(u_i, v_i)\}$. $\delta_o(p) = X$ if and only if $X = \cup(u_i, v_i)$.*

It is well-known that in $\mathbb{R}$ every open set can be expressed as union of (rational) open intervals. Thus δ_o is a surjection. This also induces a representation δ_c for closed sets by simply looking at complement.

δ_o has several desirable properties that aid us in investigating $\mathbb{R}$ under ρ_C:

Theorem 4.11.

1. *X is δ_o-computable $\iff$ X is ρ_C-r.e.*
2. *Union and intersection are δ_o-computable.*
3. *For a continuous function f, the inverse mapping $f^{-1}(X)$ defined on open subsets is δ_o-computable if f is ρ_C-computable*

We shall provide some examples of computable open (closed) sets and comment on their ρ_C property.

Example 4.12

1. *A closed interval $[a, b]$ is recursive if a, b are computable.*
2. *A closed ball $B_a(r)$ is recursive if a, r are both computable.*
3. *Same can be said of open sets.*
4. *Open sets $\{(x, y) \in \mathbb{R}^2 | x < y\}$ are recursive.*
5. *The set of all computable real number U_N is r.e. open.*

Lastly, we consider various representations of compact subsets K of $\mathbb{R}$.

Definition 4.13. *Consider the following representations of K, given δ_o.*

1. *(closed and bounded)*

   κ_{cb}, which takes a representation q of open sets together with a rational number r. $\kappa_{cb}(r, q) = X$ if and only if $X = \mathbb{R} \setminus \delta_o(q)$ and $X \subset [-r, r]$. (Note that surjectivity is provided by Henie-Borel theorem.)

2. *(weak covering)*

   *κ_w, which takes $\{w_i\}$, sequence of notations for (rational) **finite open covers**. $\kappa_w(p) = X$ if and only if p is exactly all open covers of X.*

3. *(strong covering)*

   κ, defined similar to κ_w, but p is only allowed to contain 'minimal' open covers, i.e. for open cover $\cup I_i$ of X, $X \cup I_i \neq \emptyset$ for all I_i.

κ is useful because of the following properties:

Theorem 4.14. *Let κ be the strong covering representation.*

1. *A compact set is recursive if and only if it is κ-computable.*

2. *By the extreme value theorem, we may define a maximum function* $\max : K \to \mathbb{R}$ *on compact sets. It is* (κ, ρ_C)*-computable.*

We can also show reducibility relations between the three representations. It can be shown[4] that $\kappa < \kappa_w$. The following relation is more curious:

Theorem 4.15. $\kappa_w \equiv \kappa_{cb}$

It says the closed bounded representation is equivalent to the (weak) open cover representation. For this reason it is aptly named **computational Heine-Borel theorem**.

5 Further Studies

5.1 Conclusion

We have come a long way to introduce computability on abstract objects, while relating it to topological structures. We started by introducing computability on $\mathbb{N}$ and then naturally expand it to $\mathbb{N}^{\mathbb{N}}$ so that any abstract object of continuum cardinality can be investigated for its computability. However, we recognize that such investigation is most fruitful when it is compatible with a specific topology. This is especially true for $\mathbb{R}$, whose topological structure allows us to investigate its computability in the familiar framework of standard real analysis.

Our approach, despite being an elementary one, provides numerous ramifications for further investigations and a solid foundation for such. We shall outline some of those potential topics here.

5.2 Computable Analysis

A natural next step is applying computability to all kinds of familiar topics in analysis. Specifically, one may consider how the space of continuous functions may be represented and how the notion of computability may be extended. One could also study the computability with respect to differentiation and integration. To this end, the readers are directed to the latter chapters of [16]. Further, the computational complexity of real numbers can be studied by looking at both the look-ahead (the length of input to be examined for a given length of output) and the number of computations. However, the usual generalization seems to yield very little worthy results, and some different approaches are detailed in [17, 16].

5.3 Dynamical System

Complex objects such as Mandelbrot set and Julia set are examples of dynamical system, which exhibits complex behaviors defined by simple rules. They can be studied in relation to the computability just defined. For a more recent account of that interplay, see [7, 6]. The computability of Julia set has been investigated quite extensively in [5], and questions on complexity are asked in [12]. It can be proved that the Mandelbrot set is co-r.e.[10], but the broader aspects of it recursiveness are still unknown.

5.4 More Topological Structures

Our investigation in subsets of $\mathbb{R}$ can be generalized to arbitrary metric spaces[14, 3]. Important topological properties such as metrizability[9] and the separation axioms[18] can be studied in the framework of computability. Further, the concept of computability can be extended to other topological structures of interest. For example, *computable manifolds* are discussed in [11].

5.5 Alternative Models of Computability

There are other non equivalent ways of defining computability on real numbers. The real RAM model, for example, is analogous to the integer RAM model we proposed for basic computability, but allowed to hold real numbers in memory. It is used as the standard computability concept in a number of literatures. It is much more powerful than our finite precision model of computability but cannot be realized in any way. For other models, including those introduced by Banach/Mazur and Pour-El/Richards, a brief introduction can be found in the last chapter of [16].

References

[1] Beyer, W. Mandelbrot set. `https://en.wikipedia.org/wiki/File:Mandel_zoom_00_mandelbrot_set.jpg`, 2013.

[2] Brattka, V. Computability over topological structures. In *Computability and Models*. Springer, 2003, pp. 93–136.

[3] Brattka, V., and Presser, G. Computability on subsets of metric spaces. *Theoretical Computer Science 305*, 1-3 (2003), 43–76.

[4] Brattka, V., and Weihrauch, K. Computability on subsets of euclidean space i: Closed and compact subsets. *Theoretical Computer Science 219*, 1-2 (1999), 65–93.

[5] Braverman, M., and Yampolsky, M. *Computability of Julia Sets*. Springer, Berlin, Heidelberg, 2009.

[6] Buescu, J., Graça, D. S., and Zhong, N. Computability and dynamical systems. In *Dynamics, Games and Science I*. Springer, Berlin, Heidelberg, 2011, pp. 169–181.

[7] Collins, P. J. *Computable analysis with applications to dynamic systems*. Stichting Centrum voor Wiskunde en Informatica, 2010.

[8] Cutland, N. *Computability: An introduction to recursive function theory*. Cambridge University Press, 1980.

[9] Grubba, T., Schröder, M., and Weihrauch, K. Computable metrization. *Mathematical Logic Quarterly 53*, 4-5 (2007), 381–395.

[10] Hertling, P. s the mandelbrot set computable? *Mathematical Logic Quarterly 51*, 1 (2005), 5–18.

[11] Iljazović, Z., and Validžić, L. Computable neighbourhoods of points in semi-computable manifolds. *Annals of Pure and Applied Logic 168*, 4 (2017), 840–859.

[12] Rettinger, R., and Weihrauch, K. The computational complexity of some julia sets. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing* (2003).

[13] Turing, A. M. On computable numbers, with an application to the entscheidungsproblem. *J. of Math 58* (1936), 345–363.

[14] Weihrauch, K. Computability on computable metric spaces. *Theoretical Computer Science 113*, 2 (1993), 191–210.

[15] Weihrauch, K. *A simple introduction to computable analysis*. Fernuniv., Fachbereich Informatik, 1995.

[16] Weihrauch, K. *Computable analysis: an introduction*. Springer Science & Business Media, 2000.

[17] Weihrauch, K. Computational complexity on computable metric spaces. *Mathematical Logic Quarterly 49*, 1 (2003), 3–21.

[18] Weihrauch, K. Computable separation in topology, from t0 to t2. *J. Univers. Comput. Sci. 16*, 18 (2010), 2733–2753.

[19] Weihrauch, K. *Computability*, vol. 9. Springer Science & Business Media, 2012.