Type 2 Recursion Theory

May 7, 2025

Abstract

The type 1 recursion theory, which is the computability on denumerable sets such as $\mathbb{N}$, is well established. In this we explore what we refer to as type 2 recursion theory, which is the computability on sets with the cardinality of the continuum, like the set of reals numbers $\mathbb{R}$. For instance, the set of infinite words over a finite alphabet Σ with at least two elements (commonly 0 and 1), has this cardinality.

In this document, we aim to reformulate some of Weihrauch's works ([Wei00]) in order to develop another way of understanding them. Our focus is on the prerequisites needed to discuss complete problems for sets of the size of $\mathbb{R}$, with respect to a given reduction.

Contents

1 Preliminaries

1.1 State of the Art

Even though models of computation have existed way before the twentieth century, the automation capability of modern computers made answering questions such as "can we design a machine that proves every mathematical statement" central in Science. That's a reason that justify why computability grew so much during the second half of the 20th century. Since in practice, we can only execute a machine for a finite amount of time, it is natural that early considerations focused on discrete objects, such as the set $\mathbb{N}$ or functions $\mathbb{N} \to \mathbb{N}$.

About the choice of a model of computation, the Turing-Church Thesis states that the computability with the Turing Machine correspond to our intuitive concept of what is computable. Although others models such has λ-calculus exists, in computability, the Turing Machine is usually chosen for its simplicity.

Definition 1.1 ((Type 1) Turing Machine). *A Turing machine is an abstract computation model made of:*

- *Three tapes that each contains an infinite number of cells. Each cell contain exactly one symbol of a finite alphabet $\mathcal{A}$ and is said initialized with the blank symbol $B \in \mathcal{A}$. Additionally, each tape has a head that moves from cell to cell (to rush the next one, go right and to rush the previous one, go left), which may write a symbol $s \in \mathcal{A} \setminus \{B\}$ on its current cell, or read the current symbol contained. Heads are initially above the first cell of each tape, and logically a head above the first cell of a tape can't move to left.*
  - *The first tape is called the **entry tape**, since it contain a word which is left to right "read-only" (i.e. its head can only move to right, and read the current symbol on its cell : it can't write on it).*
  - *The second tape is the **working tape**, which is "read-and-write".*
  - *The third one is the **output tape**, and is left to right "write-only".*
- *A finite number of **states**, including the starting state and the finite state.*
- *A **transition table**, which given the symbols below lecture heads and the current state, returns the next state. The finite state doesn't have a next state.* ∘

Before the computation, the TM may contain (on the entry tape) a finite word x over the alphabet $\mathcal{A} \setminus \{B\}$ which is called the entry.

Such a machine aims to use the working tape to, given a finite entry written in the entry tape to compute a finite output and write it on output tape.

We denote $M(x) \downarrow = y$ if the Turing Machine M with x as entry stops (i.e. reach the finite state) with the output y (what is written in the output tape).

Definition 1.2 (Computability). *We say that a Turing Machine M compute the function $f : X \to Y$ if*

$$\forall x \in X,\ M(x) \downarrow = f(x)$$

where X and Y are countably infinite sets. In this case, f is called computable. ∘

X and Y can't be bigger than countably infinite because its elements are represented in the entry tape as finite sequences of symbols that belongs to a finite set ($\mathcal{A}$).

Such functions are called (type 1) recursive functions.

Some functions aren't recursive (not computable with TM). The canonical example is the Halting Problem : $\{< e, x > \in \mathbb{N} : \Phi_e(x) \downarrow\}$.

1.2 Representation of Σ^ω

Definition 1.3 (Type 2 Turing Machine). *A Type 2 Turing Machine is similar to a Type 1 TM, with the difference that it works with "type 2" (infinite words, on Σ^ω) numbers, i.e. it is defined with three tapes:*

- *A read-only infinite entry tape, with an infinite word written on it.*
- *A read and write working tape.*
- *A write-only output tape. If the TM accepts the entry (i.e. it is on his definition domain), the machine will write an infinite word and never stop. Otherwise, it will stop.* ∘

Thus, a Type 2 TM define a function $f :\subseteq \Sigma^\omega \to \Sigma^\omega$.

Definition 1.4. *A real number is called **computable** if there exist a type 2 Turing machine that can produce its decimal expansion.* ∘

For example, 1 is computed by a TM that writes $1.00\ldots$ with an infinite number of zeros, or $0.99\ldots$ with an infinite number of nines, with the alphabet $\Sigma_{10} = [9] \cup$ ".". There is no unique way to represent a real number. We may represent reals with the base10, which correspond to a representation $\rho_{10} : \Sigma_{10}^\omega \to \mathbb{R}$.

But we will show that with such a representation, even very simple functions like multiplication by 3 isn't computable on a Type 2 TM.

Theorem 1.5 (Vasco Brattka [BHW08]). *With the representation ρ_{10}, the multiplication by 3*

$$\text{mult3} : x \mapsto 3 * x$$

isn't computable (computability on type 2 TM will be seen later).

Proof. Suppose that we have such a machine M and let $p = 0.333\ldots$ with an infinite number of threes. We know that M should start writing on the output tape before reading the entire entry (because the input is infinite, and that at every state of computation, only a finite time happened since the beginning of the execution), i.e. a finite number of symbols on input tape has been read before writing the first symbol on output tape. Thus, for the following real

$$q = 0.333\ldots343\ldots$$

with enough threes before the four, M starts by writing $0.99\ldots$ on the output tape.

But, in this case, the number computed by $M(q)$ cannot be strictly greater than 1 (we can't erase what we have written in output tape since it's one-way). This leads to a contradiction because since $M(p) = 1$ and $p < q$, we should have $M(p) < M(q)$. ∎

Thus, we should look for another representation, that would "varies more smoothly". That's what the Cauchy representation allows.

Notation (Cauchy representation)

The Cauchy representation $\rho_C : \Sigma^\omega \to \mathbb{R}$ is defined by :

$$\rho_C(w_0\#w_1\#\ldots\#w_n\#\ldots) = x \leftrightarrow \lim_{n\to\infty} \rho_{\mathbb{Q}}(w_n) = x \wedge \forall i, |\rho_{\mathbb{Q}}(w_{i+1}) - \rho_{\mathbb{Q}}(w_i)| \leq 2^{-i}$$

where $\rho_{\mathbb{Q}} : \Sigma^{<\mathbb{N}} \to \mathbb{Q}$ is a representation for rational numbers.

Theorem 1.6 (Vasco Brattka [BHW08]). *The usual functions on reals are computable.*

Proof (For the multiplication). Let's show that

$$\begin{aligned}\text{mult}\colon \mathbb{R} * \mathbb{R} &\to \mathbb{R}\\ : (x, y) &\mapsto x \times y\end{aligned}$$

is computable.

Suppose that we have $(p_i)_{i\in\mathbb{N}}$ and $(q_i)_{i\in\mathbb{N}}$ that converges rapidly respectively to x and y.

The multiplication of rational numbers is computable. Thus, we can compute $(r_i)_{i\in\mathbb{N}}$ where $r_i = p_{i+1} \times q_{i+1}$.

$$\begin{aligned}|r_{i+1} - r_i| &= |(p_{i+2} \times q_{i+2}) - (p_{i+1} \times q_{i+1})|\\ &= |(p_{i+2} - p_{i+1})q_{i+2} + p_{i+1}(q_{i+2} - q_{i+1})|\\ &\leq |p_{i+2} - p_{i+1})||q_{i+2}| + |p_{i+1}||q_{i+2} - q_{i+1})|\\ &\leq 2^{-(i+1)} + 2^{-(i+1)} && \text{since } |q_{i+2}| \leq 1 \text{ and } |p_{i+1}| \leq 1\\ &\leq 2^{-i}\end{aligned}$$

Thus, $\lim_{i\to\infty} r_i = x \times y = \text{mult}(x, y)$, and (r_i) converges rapidly. ∎

1.3 Cantor Topology

Let's describe a topology, that will allow us later to talk about continuity.

Proposition 1.7 (Cantor Space). *The Cantor set $2^{\mathbb{N}}$ with the metric*

$$\mathrm{d_C}(p, q) = \begin{cases} 2^{-\min\{i\in\mathbb{N}:p[i]\neq p[i]\}} \text{ if } p \neq q\\ 0 \text{ otherwise}\end{cases}$$

is a metric space.

Proof.

- $\mathrm{d_C}$ is **symmetric** since $\mathrm{d_C}(p, q) = \mathrm{d_C}(q, p)$

- Let $p,\ q \in 2^{\mathbb{N}}$. $p = q \implies \mathrm{d_C}(p, q) = 0$. In the other side, if $\mathrm{d_C}(p, q) = 0$ there exist no $i \in \mathbb{N}$ such that $p[i] \neq q[i]$. Thus, $p[i] = q[i]$ for all i, i.e. $p = q$.
  Hence, $\mathrm{d_C}$ verify **separation**.

- Let $p,\ q,\ r \in 2^{\mathbb{N}}$. Let's show that the Triangular Inequality (T.I) is satisfied :

  $$\mathrm{d_C}(p,\ q) \leq \mathrm{d_C}(p,\ r) + \mathrm{d_C}(q,\ r)$$

  We note $i = \mathrm{d_C}(p,\ q)$, $j = \mathrm{d_C}(p,\ r)$ and $j' = \mathrm{d_C}(q,\ r)$.

  - If $i = 0$, then the T.I is satified, since j and j' are positive.
  - If $j = 0$, then $p = r$ and the inequality became $\mathrm{d_C}(p,\ q) \leq \mathrm{d_C}(q,\ r)$ which is true because $\mathrm{d_C}$ is symmetric
  - If $j' = 0$, then we conclude by symmetry of role played by q and r.

  Let's now suppose that $i,\ j,\ j'$ are strictly greater than 0. By contradiction, suppose that $i > j + j'$.

– If $j < j'$, then

$$p[j] \neq r[j], \text{ by definition of j} \tag{1}$$
$$p[j] = q[j], \text{ because } i > j \tag{2}$$
$$q[j] = r[j], \text{ because } j' > j \tag{3}$$

This lead to a contradiction : $(2) \wedge (3) \implies \neg(1)$.

– If $j > j'$, then we conclude as the previous point, by symmetry of role played by q and r.

– If $j = j'$, then we have $p[j] \neq r[j]$ and $q[j'] = q[j] \neq r[j'] = r[j]$. Thus, $p[j] \neq q[j]$. But $j < i$ implies that $p[j] = q[j]$. contradiction. ∎

Thus, the Cantor space induces a topological space $(2^{\mathbb{N}}, \mathcal{T}_{\mathrm{C}})$. It is complete and totally disconnected. Additionally, we can talk about topology for every set Σ that is finite with at least two elements, because $\Sigma^{\mathbb{N}}$ would be in bijection with the Cantor set. We note Σ^{ω} instead of $\Sigma^{\mathbb{N}}$, and we also call cantor space every $(\Sigma^{\omega}, \mathcal{T}_{\mathrm{C}})$ for such a set Σ.

In such spaces, the open balls are the cylinders :

Notation (Cylinder)

We call cylinder generated by a finite word $w \in \Sigma^{<\omega}$ the set of infinite next words starting by w :

$$w\Sigma^{\omega} = \{p \in \Sigma^{\omega} : \forall i < |w|,\ p[i] = w[i]\}$$

Thus, we can define open sets.

Definition 1.8 (Open set). *The open sets of a cantor space $(\Sigma^{\omega}, \mathcal{T}_{\mathrm{C}})$ are the (arbitrary) unions of cylinder, i.e. a set O is open if there exists $W \subseteq \sigma^{<\omega}$ such that:*

$$O = \cup_{w \in W} w\Sigma^{\omega}.$$

Closed sets are complementary of open sets. ∘

Proposition 1.9 (open-closed sets). *Finite unions of cylinders are open and closed sets.*

Proof. Let $O = \bigcup_{w \in W} w\Sigma^{\omega}$ be an open set where $|w|$ is finite. Let

$$W' = \{w' \in \Sigma^{<\mathbb{N}} : \forall w \in W,\ \neg(w' \prec w \vee w \prec w')\}.$$

Show that $C = \bigcup_{w' \in W'} w'\Sigma^{\omega}$ is complementary of O.

Let $p' \in \Sigma^{\omega} \setminus O$. Since W is finite, there is a size $n \in \mathbb{N}$ such that $\forall w \in W,\ n > |w|$. Thus, any finite prefix w' of p' that is at most size n satisfies:

- w' is not a prefix of any $w \in W$ by a length argument, i.e. $\forall w \in W,\ \neg(w' \prec w)$.
- w' has no prefix in W, i.e. $\forall w \in W,\ \neg(w \prec w')$ because otherwise p' would be in O.

Hence, we can conclude that p' has a prefix in W', by definition of W', and $O^{\complement} = C$. Additionally, C is an open set by definition, which allow us to finish the proof. ∎

2 Computability on reals and Continuity

We could represent Type 2 TM with a finite code, what is usually done with classical TM, but here we would rather represent the machine by an infinite representation of the function computed by it (see definition 3.1). Since by definition, the function computed by a machine M is computable, and that computability implies continuity (see proposition 2.6), such a representation always exist.

In the case of functions on Σ^ω, we can define computability and continuity as following.

Definition 2.1 (Σ^ω-computability). *A function $f :\subseteq \Sigma^\omega \to \Sigma^\omega$ is called **Σ^ω-computable** (or simply **computable**) if it is computed by a Type 2 Turing Machine.* ◦

Definition 2.2 (Σ^ω-continuity). *A function $f :\subseteq \Sigma^\omega \to \Sigma^\omega$ is called **Σ^ω-continuous** (or simply **continuous**) if for any finite prefix w of $f(p)$, there exists a finite prefix v of p s.t :*

$$f(v\Sigma^\omega \cap \mathrm{dom}(f)) \subseteq w\Sigma^\omega.$$ ◦

Here, $v\Sigma^\omega$ and $w\Sigma^\omega$ are open sets.

To works with more general functions, defined on arbitrary sets, we introduce the representation of a set.

Definition 2.3 (Representation of a set). *A representation of a set X is a surjective function $\delta :\subseteq \Sigma^\omega \to X$.* ◦

We will say that a general function f is continuous (or computable) depending on a continuous (resp. computable) Σ^ω-function F that satisfies a property on representation, which can be described by a commutative diagram :

$$\begin{array}{ccc} \Sigma^\omega & \xrightarrow{F} & \Sigma^\omega \\ \downarrow{\scriptstyle \delta_X} & & \downarrow{\scriptstyle \delta_Y} \\ X & \xrightarrow{f} & Y \end{array}$$

A such function will be named a realizer.

Definition 2.4 (Realizer). *Let $f : X \to Y$ be a function and $\delta_X : \Sigma^\omega \to X$ and $\delta_Y : \Sigma^\omega \to Y$ be representations.*
*A function $F :\subseteq \Sigma^\omega \to \Sigma^\omega$ is called **a realizer** of f if :*

$$\delta_Y F(p) = f\delta_X(p), \text{ for all name } p \in \mathrm{dom}(\delta_X),$$

i.e., such that every δ_X-name of some $x \in X$, the value $F(p)$ is a δ_Y-name for $f(x)$. ◦

Definition 2.5 ((δ_X, δ_Y)-computability, (δ_X, δ_Y)-continuity). *Let (X, δ_X) and (Y, δ_Y) be sets with representations. A function $f : X \to Y$ is called (δ_X, δ_Y)-computable (resp. (δ_X, δ_Y)-continuous) if there exist a **computable** (resp. **continuous**) realizer of f.* ◦

We denote $\mathcal{C}(X, Y)$ the set of **total** continuous functions from X to Y.

Proposition 2.6 (Weihrauch, 2000). *Any computable function $F :\subseteq \Sigma^\omega \to \Sigma^\omega$ is continuous.*

It also hold for functions $F : \mathbb{R} \to \mathbb{R}$. More generally, we have the following property.

Proposition 2.7. *For any function $f : X \to Y$ that is representable, there exist representations δ_X and δ_Y such that f is $(\delta_X, \delta_Y) - continuous$.*

Theorem 2.8. *Any function $F : \Sigma^\omega \to \Sigma^\omega$ is computable on a type 2 TM with oracle iff it is continuous.*

Hint : take as oracle the representation of the function.

3 Representation of $[\Sigma^\omega \to \Sigma^\omega]$

Definition 3.1 (Representation of continuous function). *We represent a continuous function $F : \Sigma^\omega \to \Sigma^\omega$ as a list δ_f of couples s.t for all finite prefix w of each $p \in \Sigma^\omega$, there exist a finite prefix v of p such that $(w, v) \in \delta_f$ that satisfies the definition of continuity, i.e*

$$\delta_f = \{(w, v) : \exists (p, w', v') \in (\Sigma^\omega)^3, ww' = f(p) \wedge vv' = p \wedge f(v\Sigma^\omega) \subseteq w\Sigma^\omega\} \qquad \circ$$

By encoding δ_X, δ_Y and a realizer, we should be able to represent a more general function (δ_X, δ_Y)-continuous function $f : X \to Y$.

Notation (Set of functions)

We note $[X \to Y]_{\delta_X, \delta_Y}$ the set of total **continuous** functions from X to Y.

A representation of $[\Sigma^\omega \to \Sigma^\omega]$ would be a surjective function that given an infinite word, returns a total continuous function from $\Sigma^\omega \to \Sigma^\omega$.

We can define Type 2 TM with n arguments by considering that the entry tape alternatively contain one symbol of each argument.

Definition 3.2 (Universal Type 2 TM). *We define $\Gamma_u : \Sigma^\omega \to \Sigma^\omega$ as the function computed by the Type 2 TM described by Algorithm 1.*
Thus, we can define the representation

$$\Psi : \Sigma^\omega \to [\Sigma^\omega \to \Sigma^\omega]$$
$$\Psi(\delta)(p) := \Gamma_u \langle \delta, p \rangle.$$

Algorithm 1: Universal Turing Machine Γ_u

Data: $\langle \delta, p \rangle \in \Sigma^\omega$ where :
δ : an enumerable list of couples of prefixes $\in \Sigma^{<\omega}$ that represent a function f
p : an infinite word
Result: If $p \in dom(f)$, returns the **infinite word** $f(p)$, else returns a **finite word**.

```
1 for k = 1 → ∞ do
2 |   if ∃(w, v) ∈ δ, |w| = k and v is a prefix of p then
3 |   |   write w on output tape // we only write missing symbols, by storing for
  |   |       example the length of current prefix written in output tape
4 |   end
5 end
```

That's the only code used by Weihrauch [Wei00].

Conclusion

Finally, we have observed that topological consideration have led us to useful representations for Σ^ω and $[\Sigma^\omega \to \Sigma^\omega]$, enabling us to capture a notion of computability that is coherent and compatible with common functions on the real numbers.

Now, we have to choose an appropriate type of reduction (Cook-Turing reduction, Karp reduction ..) and attempt to characterize some computability classes. Do they have complete problem ? Are there multiple (or even infinitely many) degrees of unsolvability ? Is there a complete problem for each degree ?

References

[BHW08] Vasco Brattka, Peter Hertling, and Klaus Weihrauch. *A Tutorial on Computable Analysis*, pages 425–491. Springer New York, New York, NY, 2008.

[Wei00] Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.