Recap

Classically, computability is introduced for functions $f :\subseteq (\Sigma_1^*)^n \to \Sigma_1^*$ where Σ_1 is some alphabet, for example by means of Turing machines. For computing functions on other sets such as the rational numbers or finite graphs, words are used as names of elements of M. Under this view, a machine transforms words to words without understanding the meaning given to them.

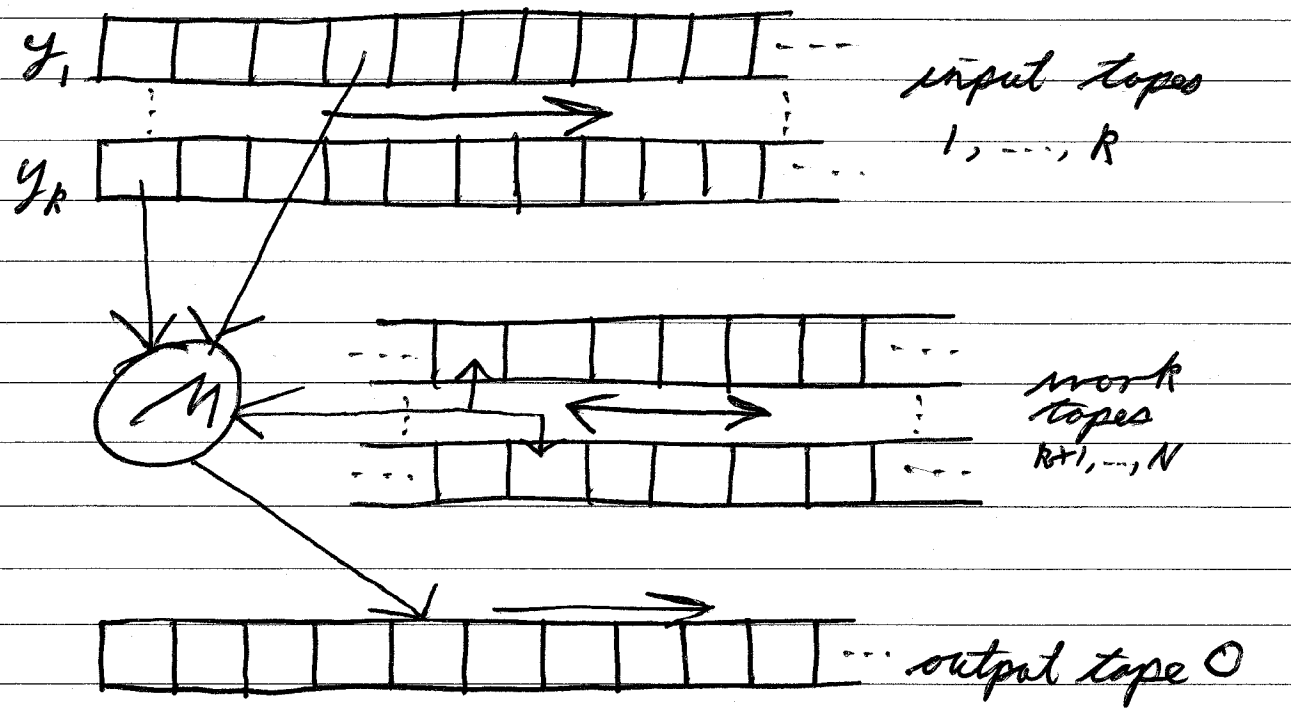
Equivalently, one can start with computable functions on the natural numbers, instead of words, and use numbers as names. Since Σ_1^* is only countable, this method cannot be applied for introducing computability on uncountable sets M like $\mathbb{R}$, $\mathcal{O}(\mathbb{R})$, $2^{\mathbb{R}}$, $C[0,1]$.

We extend the above concepts by using infinite sequences of symbols as names, and by defining computability for functions which transform such infinite sequences. The set Σ_1^ω (where $|\Sigma_1| \geq 2$, finite) has the same cardinality as $\mathbb{R}$, $\mathcal{O}(\mathbb{R})$, $C[0,1]$, etc, so it can serve as a set of names for every set with at most the continuum cardinality.

For convenience, we will assume Σ_1 is a fixed alphabet containing 0 and 1.

Type-1 Machines

There are many equivalent definitions of a Turing Machine. We will be interested in Turing Machines with k input tapes, finitely many work tapes, and an output tape. The input and output tapes will be one-way, and the machine must read from the input tapes in order (that is, it cannot "go back" or write), and it must write in order to the output.



Formally, a Turing Machine is given by an input/output alphabet Σ, with a special symbol $B \notin \Sigma$, a work alphabet T s.t. $\Sigma \cup \{B\} \subseteq T$, a number k of input tapes, and a "flowchart" (relation) operating on the tapes.

The above diagram computes $y_0 = f_M(y_1, \ldots, y_k)$.

Type-2 Machines

A Type-2 Machine is a Turing Machine with k input tapes together with a type specification $(Y_1, \ldots, Y_k, Y_0)$ with $Y_i \in \{\Sigma_i^{1*}, \Sigma_i^{\omega}\}$, giving the type for each input tape and the output tape.

We can now define the string function $f_M :\subseteq Y_1 \times \cdots \times Y_k \to Y_0$ computed by a Type-2 Machine M. The initial tape configuration for input $(y_1, \ldots, y_k) \in Y_1 \times \cdots \times Y_k$ is as follows: for each input tape i, the sequence $y_i \in Y_i$ (not necessarily finite) is placed on the tape immediately to the right of the head, all other tape cells contain the symbol B. On all other tapes, all tape cells contain the symbol B.

For all $y_0 \in Y_0, y_1 \in Y_1, \ldots, y_k \in Y_k$, we define:

(1) Case $Y_0 = \Sigma_1^{1*}$: $f_M(y_1, \ldots, y_k) := y_0 \in \Sigma_1^{1*}$ iff M halts on input $(y_1, \ldots, y_k)$ with y_0 on the output tape.

(2) Case $Y_0 = \Sigma_1^{\omega}$: $f_M(y_1, \ldots, y_k) := y_0 \in \Sigma_1^{\omega}$ iff M computes forever on input $(y_1, \ldots, y_k)$ and writes y_0 on the output tape.

We call a string function $f :\subseteq Y_1 \times \cdots \times Y_k \to Y_0$ computable iff it is computed by some Type-2 Machine M.

Important Remarks

Notice that $\varphi_M(y_1, \ldots, y_k)$ is undefined if M computes forever but only writes finitely many symbols on the output tape. We do not use the partial results of such computations in our semantics.

Type-2 machines are clearly as realistic and as powerful as Type-1. Infinite inputs or outputs do not exist and infinite computations cannot be finished in reality, but finite computations on finite initial parts of inputs producing finite initial parts of outputs can be realized on physical devices as long as enough time and memory are available. That is, we can always approximate (using digital computers or otherwise) by finite physical computations with arbitrary precision.

The restriction to one-way output guarantees that any partial output of a finite initial part of a computation cannot be erased in the future, thus is "final". For this reason, models of computation with two-way output would not be very useful.

Computable Elements

We now define computable elements of Σ_1^{1*} and $\Sigma_1^{1\omega}$ straightforwardly:

(1) Every word $w \in \Sigma_1^{1*}$ is computable.

(2) A sequence $p \in \Sigma_1^{1\omega}$ is computable iff the constant function $f : \{0\} \to \Sigma_1^{1\omega}$ $f() = p$ is computable.

(3) A tuple $(y_1, y_2, \ldots, y_k)$ is computable iff each component y_i is computable.

Eg 1 A sequence $p \in \Sigma_1^{1\omega}$ is computable iff $p = f(\lambda)$ for some computable function $f : \Sigma_1^{1*} \to \Sigma_1^{1\omega}$.

Eg 2 A constant function $f : Y_1 \times \cdots \times Y_k \to Y_0$ is computable iff its value $c \in Y_0$ is computable.

Eg 3 Every projection $\pi_i : Y_1 \times \cdots \times Y_k \to Y_i$ is computable.

Eg 4 Define $f :\subseteq \Sigma_1^{1\omega} \to \Sigma_1^{1*}$ by:

$$f(p) := \begin{cases} 1 & \text{if } p \neq 0^\omega \\ \perp & \text{otherwise.} \end{cases}$$

Clearly f is computable, but if we replace $\perp$ with 0, it is not computable.

(2.6)

Decimal Fractions Revisited

No Type-2 machine multiplies infinite decimal fractions by 3.

Suppose that M computed $x \mapsto 3 \cdot x$, then it must operate correctly on the name $0.333\ldots$ of $1/3$ as input. The output must be $0.999\ldots$ or $1.000\ldots$.

Consider the first case. Let k be the number of steps which M operates before writing the prefix "0." on the output tape. During this time, the machine reads only finitely many symbols, say the prefix "$0.w$" from the input tape.

Consider now the input sequence $0.w999\ldots$ which is the name of a real number strictly larger than $1/3$. But since 3-times such a number is strictly larger than 1, then M does not behave correctly on this input!

In the second case, the argument is similar. Take $0.w000\ldots$.

A computability concept under which multiplication by a constant is not computable is not very useful!

Characterization on Σ_1^*

Def For any partial function $h :\subseteq \Sigma_1^* \to \Sigma_1^*$ with prefix-free domain, define $h_* : \Sigma_1^\omega \to \Sigma_1^*$ by:

$$h_*(p) := \begin{cases} h(w) & \text{if } w \sqsubseteq p \wedge w \in \mathrm{dom}(h) \\ \perp & \text{if } w \not\sqsubseteq p \ \ \forall w \in \mathrm{dom}(h). \end{cases}$$

Lem A function $f :\subseteq \Sigma_1^\omega \to \Sigma_1^*$ is computable iff $f = h_*$ for some computable function $h :\subseteq \Sigma_1^* \to \Sigma_1^*$ with prefix-free domain.

Proof: Let M be a Type-2 machine computing $f :\subseteq \Sigma_1^\omega \to \Sigma_1^*$. Define $h : \Sigma_1^* \to \Sigma_1^*$ by

$$h(w) := \begin{cases} f(w0^\omega) \text{ if} & \text{on input } w0^\omega, M \\ & \text{halts after } |w| \text{ steps} \\ \perp & \text{otherwise} \end{cases}$$

Then h is computable, has prefix-free domain, and satisfies $f = h_*$.

On the other hand, let M be a Turing Machine computing $h :\subseteq \Sigma_1^* \to \Sigma_1^*$ with prefix-free domain. There is a Type-2 machine N which on input $p \in \Sigma_1^\omega$ searches for the smallest $k = \langle i, t \rangle$ such that M halts on $p\langle i$ in t steps, printing $h(p\langle i)$. Since $\mathrm{dom}(h)$ is prefix-free, there is at most one such k, so N computes the function h_*. $\square$

Characterization of Σ_1^ω

Dfn For any monotone (wrt. $\sqsubseteq$) total function $h: \Sigma_1^* \to \Sigma_1^*$, define $h_\omega :\subseteq \Sigma_1^\omega \to \Sigma_1^\omega$ by:

$$p \in \mathrm{dom}(h_\omega) \iff (|h(p_{<i})|)_{i \in \mathbb{N}} \text{ unbounded}$$
$$h_\omega(p) := \sup_{i \in \mathbb{N}} h(p_{<i})$$

where $\sup_{i\in\mathbb{N}} h(p_{<i})$ is the $q \in \Sigma_1^\omega$ s.t. $h(p_{<i}) \sqsubseteq q$ $\forall i \in \mathbb{N}$.

Lem A function $f :\subseteq \Sigma_1^\omega \to \Sigma_1^\omega$ is computable iff $f = h_\omega$ for some computable monotone $h: \Sigma_1^* \to \Sigma_1^*$.

Proof: Let M be a Type-2 machine computing $f :\subseteq \Sigma_1^\omega \to \Sigma_1^\omega$. Define $h: \Sigma_1^* \to \Sigma_1^*$ by $h(w)$ is the word $x \in \Sigma_1^*$ which M on input $w0^\omega$ produces in $|w|$ steps. Then the function h is monotone and computable, and $f = h_\omega$.

On the other hand, let $h: \Sigma_1^* \to \Sigma_1^*$ be a computable monotone function. Then, for any $p \in \Sigma_1^\omega$ and $i \in \mathbb{N}$, $h(p_{<i}) \sqsubseteq h(p_{<i+1})$. There is a Type-2 machine N which on input $p \in \Sigma_1^\omega$ works in stages $n = 0, 1, \ldots$ as follows: in stage n, N extends the result $h(p_{<n-1})$ from stage $n-1$ to $h(p_{<n})$. Thus, the function $f :\subseteq \Sigma_1^\omega \to \Sigma_1^\omega$ computed by machine N satisfies $f = h_\omega$. □

Closure under Composition

Thm For $k, n \in \mathbb{N}$ and $X_1, \ldots, X_k, Y_1, \ldots, Y_n, Z \in \{\Sigma^*, \Sigma^\omega\}$, let $g_i :\subseteq X_1 \times \cdots \times X_k \to Y_i$, $f :\subseteq Y_1 \times \cdots \times Y_n \to Z$ be computable functions. Then the composition

$$f \circ (g_1, \ldots, g_n) :\subseteq X_1 \times \cdots \times X_k \to Z$$

is computable if $Z = \Sigma^\omega$ or $Y_i = \Sigma^*$ for all i.

Alternatively, if $Z = \Sigma^*$ and $Y_i = \Sigma^\omega$ for some i, then it has a computable extension:

$$h :\subseteq X_1 \times \cdots \times X_k \to Z$$

with $\mathrm{dom}(h) \cap \mathrm{dom}(g_1, \ldots, g_n) = \mathrm{dom}(f \circ (g_1, \ldots, g_n))$.

That is, the composition is computable if the final result is infinite or all intermediate results are finite, and has a computable extension if the final result is finite and some intermediate result is infinite.

Cor Every computable function maps computable elements to computable elements.

Proof: By definition elements are computable if $f : \{()\} \to \Sigma^\omega$, $() \mapsto p$ is. Then just apply the above theorem. □

Closure under Primitive Recursion

Thm Let $f' :\subseteq Y_1 \times \cdots \times Y_k \to Y_0$ be a computable function, and for each $a \in \Sigma_1$ let:

$$f_a :\subseteq \Sigma_1^* \times Y_0 \times Y_1 \times \cdots \times Y_k \to Y_0$$

be a computable function. Then:

$$g :\subseteq \Sigma_1^* \times Y_1 \times \cdots \times Y_k \to Y_0$$

defined by the recursion equations:

$$g(\lambda, y_1, \ldots, y_k) = f'(y_1, \ldots, y_k)$$
$$g(aw, y_1, \ldots, y_k) = f_a(w, g(w, y_1, \ldots, y_k), y_1, \ldots, y_k)$$

for all $w \in \Sigma_1^*$, $y_1 \in Y_1, \ldots, y_k \in Y_k$ and $a \in \Sigma_1$ is computable.

Church-Turing Remarks

It is not claimed that a function is computable by a physical machine iff it is computable by a Type-2 machine because there is no convincing reason to exclude machines with two-way output, or more sophisticated input/output conventions. The reason for Type-2 machines is out of practicality.

The Finiteness Property

Roughly speaking, the finiteness property says that, given a Type-2 Machine M such that $f_M :\subseteq \Sigma_1^\omega \to \Sigma_1^\omega$, every finite portion of the output $f_M(p)$ is already determined by a finite portion of the input p.

That is, if $u \in \Sigma_1^*$ with $u \sqsubseteq f_M(p)$, then on input p, the machine M writes the prefix u of the output $f_M(p)$ in t steps for some $t \in \mathbb{N}$. Within t steps, M can read not more than the prefix $w := p_{<t}$ of the input $p \in \Sigma_1^\omega$. Therefore, the output u depends only on the prefix $w \sqsubseteq p$, but not on the rest. We obtain $f_M[w\Sigma_1^\omega] \subseteq u\Sigma_1^\omega$.

We have, of course, already seen and used this finiteness property. Next, we will see that it is equivalent to continuity of f_M if we consider the Cantor topology on the set Σ_1^ω.

Since topology is a mathematical theory for studying approximation (among other things), it is not surprising that we can use it for describing the approximation of infinite sequences of symbols by finite ones, or real numbers by rational intervals.

Standard Topologies on Σ_1^{1*} and $\Sigma_1^{1\omega}$

$T_* := 2^{\Sigma_1^{1*}} = \{A \mid A \subseteq \Sigma_1^{1*}\}$ is called the discrete topology on Σ_1^{1*}. As a canonical base of T_*, we consider the set $\{\{\omega\} \mid \omega \in \Sigma_1^{1*}\}$.

$T_C := \{A\Sigma_1^{1\omega} \mid A \subseteq \Sigma_1^{1*}\}$ is called the Cantor topology on $\Sigma_1^{1\omega}$ and $(\Sigma_1^{1\omega}, T_C)$ is called the Cantor space over Σ_1. As a canonical base of T_C, we consider the set $\{W\Sigma_1^{1\omega} \mid \omega \in \Sigma_1^{1*}\}$.

To see that T_C is actually a topology on Σ_1^{1*}, for any word $w \in \Sigma_1^{1*}$,

$$w\Sigma_1^{1\omega} = \{wq \mid q \in \Sigma_1^{1\omega}\} = \{p \in \Sigma_1^{1\omega} \mid w \sqsubseteq p\} \in T_C.$$

For any words $v, w \in \Sigma_1^{1*}$, we have $w\Sigma_1^{1\omega} \cap v\Sigma_1^{1\omega} = \emptyset$ if neither w or v is a prefix of the other. Otherwise, we have $w\Sigma_1^{1*}$ or $v\Sigma_1^{1*}$. So, the set $\beta := \{w\Sigma_1^{1\omega} \mid w \in \Sigma_1^{1*}\}$ is closed under finite intersection, so is a base of some topology.

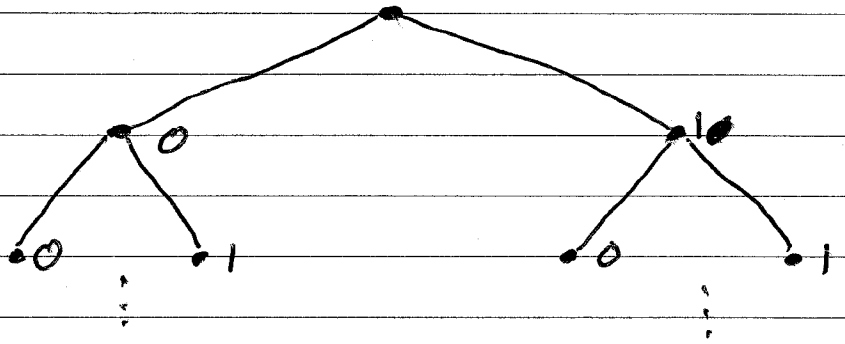
Finally, since $A\Sigma_1^{1\omega} = \bigcup\{w\Sigma_1^{1\omega} \mid w \in A\}$ for each $A \subseteq \Sigma_1^{1*}$, β must be a base of T_C.

Product topologies can be defined with the canonical base $\beta_\gamma := \{\gamma_0 \mid \gamma \in (\Sigma^{1*})^*\}$ where $\gamma = \gamma_1 \times \cdots \times \gamma_k$ $(\gamma_1, \ldots, \gamma_k \in \{\Sigma_1^{1*}, \Sigma_1^{1\omega}\})$ and $(\gamma_1, \ldots, \gamma_k) \circ \gamma := U_1 \times \cdots \times U_k$ with:

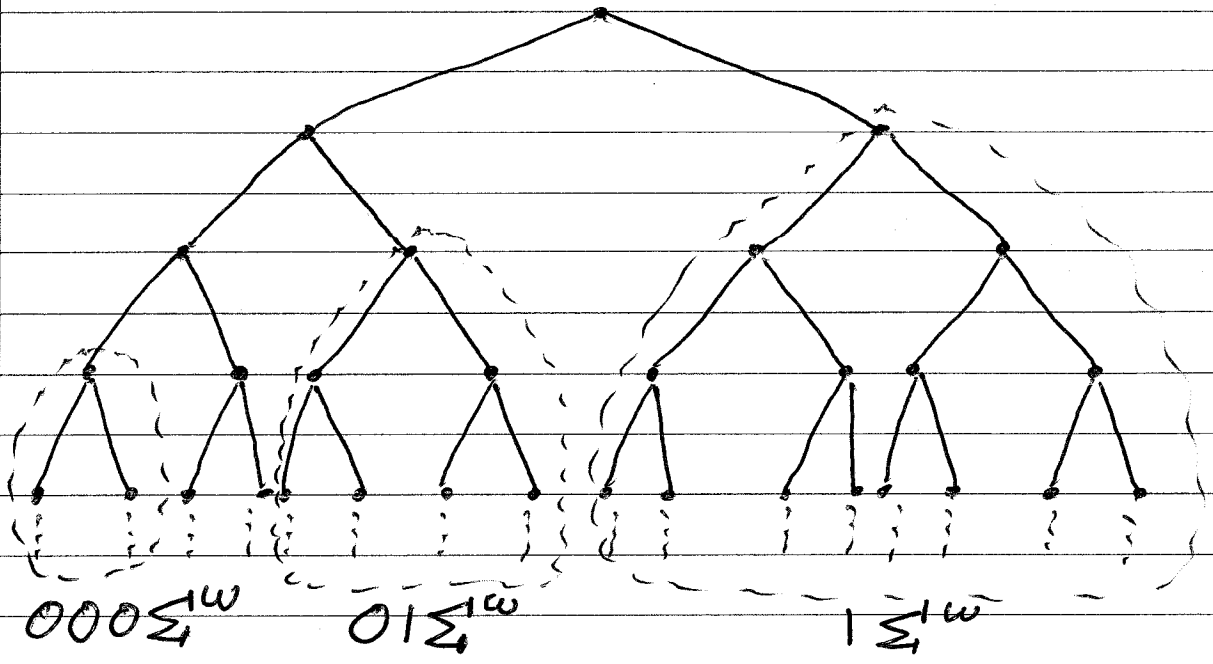
$$U_i := \begin{cases} \{\gamma_i\} & \text{if } \gamma_i = \Sigma_1^{1*} \\ \gamma_i\Sigma_1^{1\omega} & \text{if } \gamma_i = \Sigma_1^{1\omega} \end{cases}.$$

Sequences as Trees

The set Σ_1^{ω} of infinite sequences over Σ_1 can be visualized by a tree where every $\rho \in \Sigma_1^{\omega}$ is represented by an infinite descending path from the root.



A base element $w\Sigma_1^{\omega} \in T_C$ is represented by a full subtree and an open subset of Σ_1^{ω} is represented by a set of full subtrees.



The union of the 3 basic sets $000\Sigma_1^{\omega}$, $01\Sigma_1^{\omega}$, $1\Sigma_1^{\omega}$ is an open set.

Computability Implies Continuity I

The finiteness property for the total function $f_M : \Sigma_1^\omega \to \Sigma_1^\omega$ computed by a Type-2 machine M can be formulated as follows: For all $p \in \Sigma_1^\omega$, $u \in \Sigma_1^*$ with $f_M(p) \in u\Sigma_1^\omega$, there is some $w \in \Sigma_1^*$ with $p \in w\Sigma_1^\omega \subseteq f_M^{-1}[u\Sigma_1^\omega]$. That is, f_M is (T_C, T_C)-continuous.

In fact, every computable string function $f :\subseteq Y_1 \times \cdots \times Y_k \to Y_0$ is continuous.

Proof: Let M be a Type-2 machine computing f, and assume $(y_1, \ldots, y_k) \in dom(f) \subseteq Y := Y_1 \times \cdots \times Y_k$.

If $Y_0 = \Sigma_1^*$, suppose that $f(y_1, \ldots, y_k) \in \{w\}$. Since M halts on its input, it can only have read a finite prefix of each input tape of type Σ_1^ω, say u_i (define $u_i := y_i$ for those of type Σ_1^*). Then $y \in (u_1, \ldots, u_k) \circ Y$ and $f[(u_1, \ldots, u_k) \circ Y] \subseteq \{w\}$. Thus, $(u_1, \ldots, u_k) \circ Y$ is an open neighbourhood of $(y_1, \ldots, y_k)$ contained in $f^{-1}[\{w\}]$, so $f^{-1}[\{w\}]$ is open.

If $Y_0 = \Sigma_1^\omega$, suppose that $f(y_1, \ldots, y_k) \in w\Sigma_1^\omega$. To produce a finite prefix w of the output, we can only read a finite prefix of the inputs, so define u_i as before. Then $y \in (u_1, \ldots, u_k) \circ Y$ and $f[(u_1, \ldots, u_k) \circ Y] \subseteq w\Sigma_1^\omega$. But then, as before, $f^{-1}[w\Sigma_1^\omega]$ is open.

So, we have shown the preimage of the basic sets is open, so f is continuous. □

Computability Implies Continuity II

In numerous applications, functions are not computable because they are not even continuous. Continuous but not computable functions can be defined explicitly, but occur rarely in practice.

Recall that a subset $A \subseteq X$ of a topological space is called a G_δ-set iff $A = \bigcap U_i$ for some sequence of (decreasing) open sets (U_i). This is a generalization of an open set. For example, $\mathbb{Q}^c$ is a G_δ-set in $\mathbb{R}$, but $\mathbb{Q}$ is not.

The domains of computable functions have nice topological properties. If $Y = Y_1 \times \cdots \times Y_k$:

(1) $f :\subseteq Y \to \Sigma^{1*}$ computable $\Rightarrow$ $dom(f)$ open.

(2) $f :\subseteq Y \to \Sigma^{1\omega}$ computable $\Rightarrow$ $dom(f)$ is G_δ.

Proof 1: By our last proof, every $(y_1, \ldots, y_k) \in dom(f)$ has an open neighbourhood $(u_1, \ldots, u_k) \circ Y \subseteq dom(f)$. Thus, $dom(f)$ is open. □

Proof 2: Let M be a Type-2 machine which computes f. Then for each $n \in \mathbb{N}$, there is a machine M_n which on input $y \in Y$ halts iff M on input y writes at least n symbols on the output tape. So, by (1), $dom(f_{M_n})$ is open for each n, so $dom(f_M) = \bigcap dom(f_{M_n})$ is a G_δ-set. □

Cantor Spaces are Metrizable

The Cantor topology can be generated from a metric. Define $d : \Sigma_1^{\omega} \times \Sigma_1^{\omega} \to \mathbb{R}$ by $d(p,p) := 0$ and $d(p,q) = 2^{-n}$ where $n \in \mathbb{N}$ is the smallest number with $p(n) \neq q(n)$.

Moreover, Σ_1^{ω} is compact, and the basic sets are clopen balls!

Decidable Sets

We call $A \subseteq \Sigma_1^{*\omega}$ decidable iff its characteristic function is computable.

As a consequence of exactness, a subset $X \subseteq \Sigma_1^{\omega}$ is clopen iff it is decidable iff $X = A\Sigma_1^{\omega}$ for some finite set $A \subseteq \Sigma_1^{*}$.

Moreover, a total function $f : \Sigma_1^{\omega} \to \Sigma_1^{*}$ is continuous iff it is computable iff there are pairs $(u_1, v_1), \ldots, (u_R, v_R) \in \Sigma_1^{*} \times \Sigma_1^{*}$ such that:

(1) $\{u_1, \ldots, u_R\}$ is prefix-free

(2) $\Sigma_1^{\omega} = u_1\Sigma_1^{\omega} \cup \cdots \cup u_R\Sigma_1^{\omega}$

(3) $f[u_i\Sigma_1^{\omega}] = \{v_i\}$ for all i.

Closing Remarks

We have defined Type-2 Machines and computable string functions, and looked at some of their most elementary properties. We defined computable elements in terms of computability of their constant function.

A crucial observation is the finiteness property, which lead us to define the Cantor Space on Σ, a compact metric space in fact.

Another view of Cantor Spaces is to define them as those spaces that are homeomorphic to the Cantor Set. The canonical example of a Cantor Space is the countably infinite topological product of the two-point discrete space. The mapping:

$$(a_n) \mapsto \sum_1^{\infty} \frac{2a_n}{3^{n+1}}$$

is a homeomorphism from 2^{ω} onto the Cantor ~~Space~~ Set, which is defined to be the set of points lying on a line segment.

The Cantor set was discovered in 1874 by H. Smith and was introduced by G. Cantor in 1883.