Computable Analysis

Merging fundamental concepts of analysis and recursion theory, we may ask:

(1) Is the exponential function computable?

(2) Are union and intersection of closed subsets of the real plane computable?

(3) Are differentiation and integration computable operators?

(4) Is zero-finding for complex polynomials computable.

For a deep understanding and for future development of computation in analysis, a sound theoretical foundation is indispensible. Computable Analysis is developed as the theory of those functions on the real numbers (and other sets from analysis) which can be computed by machines.

We merge the concepts of limit and approximation with machine models and discrete computation.

Type-1 Theory of Effectivity

While analysis and numerical analysis have a very long tradition (Gauss and Lagrange were experts in numerical computation), it was not until the 1930s that S. Kleene, A. Church, A. Turing and others proposed various definitions of "effectively calculable" functions on the natural numbers. There is now a well-established and rich theory of computability and complexity for functions on the natural numbers or on finite words.

Although a number of authors also studied computability on the real numbers, computable analysis is still underdeveloped. Moreover, there are several non-equivalent suggestions of how to model effectivity in analysis, and in particular, computability of real functions.

~~A. Turing was the first author to introduce computable real numbers. Since that time,~~

Type-2 Theory of Effectivity

A. Turing (1936) was the first to introduce computable real numbers. Since that time, computable analysis has been developed continuously. Among the large number of publications, there are some books on computable analysis (or related topics) such as R.L. Goodstein (1959), D. Klaua (1961), S. Mazur (1963), O. Aberth (1980), B. Kushner (1984), E. Bishop and D. Bridges (1985), K. Weihrauch (1987), M. Pour-El and J. Richards (1989), J. Traub, et al (1988), K. Ko (1991), and L. Blum et al (1998).

All these books have important concepts in common, but differ in their contents and technical framework, and each author presents the topic from their individual point of view. This mirrors the fact that computable analysis has no generally accepted foundation.

K. Weihrauch (2000) presents a new coherent foundation of computable analysis, rooted in the definition of computable real functions based on the work by A. Grzegorczyk (1955) and J. ~~Hauck~~ Hauck (1973, 1978, 1980, 1981, 1982). To distinguish it from others, he called it "Type-2 Theory of Effectivity" (TTE).

Notational Conventions

We will include 0 in the natural numbers, denote by 2^A all subsets of A, and by $E(A)$ all finite subsets of A. By convention, a product of 0 sets will be $\{()\}$ where $()$ is the empty tuple.

A "correspondence" or "multivalued partial function" from A to B is a triple:

$$f = (A, B, R_f)$$

such that $R_f \subseteq A \times B$, where A is called the source, B the target, and R_f the graph of f. For $X \subseteq A$, we define the image of X under f by:

$$f[X] := \{ b \in B \mid \exists a \in X, (a,b) \in R_f \}.$$

We define the inverse of f:

$$f^{-1} := (B, A, R_f^{-1})$$

where $R_f^{-1} := \{(b,a) \mid (a,b) \in R_f\}$.

We define the range and domain of f:

$$\text{range}(f) := f[A]$$

$$\text{dom}(f) := f^{-1}[B].$$

Notation Continued

We will denote a "correspondence" f from A to B by $f :\subseteq A \rightrightarrows B$.

A "partial function" $f :\subseteq A \to B$ from A to B is a multi-valued function $f :\subseteq A \rightrightarrows B$ such that $f[\{a\}]$ contains exactly one element for each $a \in dom(f)$.

A total function $f : A \to B$ from A to B is a partial function $f :\subseteq A \to B$ such that $dom(f) = A$. The set of all total functions $f : A \to B$ is denoted by B^A.

For a partial function $f :\subseteq A \to B$, $f(a)$ denotes the single element from $f[\{a\}]$ if $a \in dom(f)$. Otherwise, $f(a) = \bot$.

For multi-valued functions $f_i :\subseteq A \rightrightarrows B_i$, define $(f_1, \ldots, f_k) :\subseteq A \rightrightarrows B_1 \times \cdots \times B_k$ by:

$$(f_1, \ldots, f_k)[\{a\}] := f_1[\{a\}] \times \cdots \times f_k[\{a\}].$$

For multi-valued functions $f :\subseteq A \rightrightarrows B$ and $g :\subseteq B \rightrightarrows C$, define the composition $g \circ f :\subseteq A \rightrightarrows C$:

$$a \in dom(g \circ f) \iff a \in dom(f) \wedge f[\{a\}] \subseteq dom(g)$$
$$(g \circ f)[\{a\}] := g[f[\{a\}]] \qquad \forall a \in dom(g \circ f).$$

$id_X : X \to X$ denotes the identity function on X.

Words and Concatenation

For any set Σ_1, Σ_1^n denotes the set of all words over Σ_1 of length n, $\Sigma_1^{\leq n}$ the set $\Sigma_1^0 \cup \cdots \cup \Sigma_1^n$, and Σ_1^* the set of all finite words over Σ_1. We denote by $|w|$ the length of word w, and by λ the word of length 0. By Σ_1^ω we denote the set $\{p \mid p : \mathbb{N} \to \Sigma_1\}$ all infinite sequences over Σ_1.

Concatenation in $\Sigma_1^* \times \Sigma_1^*$ is defined in the obvious way, and can be extended to $\Sigma_1^* \times \Sigma_1^\omega$. If $x = uvw \in \Sigma_1^*$ and $q = uvp \in \Sigma_1^\omega$, then u is a prefix of x and q ($u \sqsubseteq x$, $u \sqsubseteq q$). v is a subword of x and q ($v \triangleleft x$, $v \triangleleft q$). A is called prefix-free iff $\forall x, y \in A$, $x \not\sqsubseteq y$. By $p_{\leq n}$ we denote the prefix of length n of p. If $A, B \subseteq \Sigma_1^*$ (or $B \subseteq \Sigma_1^\omega$), then we define:

$$AB = \{up \mid u \in A,\ p \in B\}.$$

We may also write $A^n = AA \cdots A$. $\dagger *$ will be clear from the context if this means Cartesian or concatenation product.

When Σ_1 is a non-empty finite set, we may call it an alphabet, and write words over that alphabet in double quotes.

Numberings

Ordinary (Type-1) recursion theory considers the computable number functions $f :\subseteq \mathbb{N}^k \to \mathbb{N}$ and the computable word functions $g :\subseteq (\Sigma_1^*)^k \to \Sigma_1^*$. Computability can be transferred to other sets M by means of "numberings". A numbering of a set M is a surjective function $\nu :\subseteq \mathbb{N} \to M$.

For a given alphabet $\Sigma_1 = \{a_1, \ldots, a_n\}$, define the bijective standard numbering $\nu_{\Sigma_1} : \mathbb{N} \to \Sigma_1^*$ of Σ_1^* as follows:

$$\nu_{\Sigma_1}^{-1}(\lambda) := 0$$
$$\nu_{\Sigma_1}^{-1}(a_{i_k} \cdots a_{i_0}) := i_k \cdot n^k + \cdots + i_0 \cdot n^0 .$$

Then, $f :\subseteq \mathbb{N} \to \mathbb{N}$ is computable iff $\nu_{\Sigma_1} \circ f \circ \nu_{\Sigma_1}^{-1} :\subseteq \Sigma_1^* \to \Sigma_1^*$ is computable (and correspondingly for $f :\subseteq \mathbb{N}^m \to \mathbb{N}$).

The bijective Cantor pairing function $\langle , \rangle : \mathbb{N}^2 \to \mathbb{N}$ defined by $\langle x, y \rangle := y + (x+y)(x+y+1)/2$ is computable, as well as the projections $\langle \rangle_1, \langle \rangle_2$ of its inverse. For $n > 2$ arguments, we define it inductively. This will allow us to extend the definition of computable functions to functions of mixed type by calling $\nu_{\Sigma_1}, \nu_{\Sigma_1}^{-1}$ computable and closing under composition (e.g. $A \subseteq \Sigma_1^* \times \mathbb{N} \times \Sigma_1^*$).

A Sketch of TTE

Type-2 Theory of Effectivity extends ordinary (Type-1) computability and complexity theory. TTE admits two levels of effectivity: continuity, and computability as a specialization of continuity. We will present an overview of TTE informally, without using a mathematically precise model of computation.

Clearly Type-1 numberings (names) are not sufficient to name the reals, as $\mathbb{N}$ is only countable. However, real numbers can be represented by infinite sequences (for example, by infinite decimal fractions). In TTE, infinite sequences are used as names of real numbers, and machines which transfer infinite sequences to infinite sequences are used to compute (real) numbers.

$$\begin{tikzcd} {} \arrow[r, "\ldots I_2, I_1, I_0"] & \boxed{MACHINE} \arrow[r, "\ldots J_2, J_1, J_0"] & {} \end{tikzcd}$$

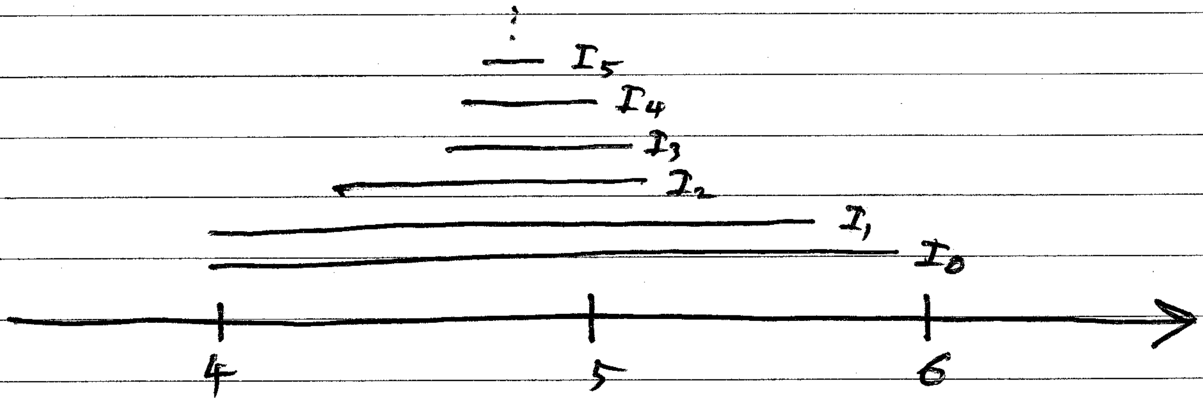
On input $(I_0, I_1, I_2, \ldots)$, from time to time, the machine reads a new sequence element I_k from its input stream, and from time to time, it writes a new sequence element J_m to its output stream. I_{k+1} must be read after I_k, and J_{m+1} must be written after J_m.

A Naming System for $\mathbb{R}$

It turns out that infinite decimal fractions induce a computability concept that is not very interesting. Instead, we can use infinite sequences of nested intervals as names.

For now, we define a name of a real number $x \in \mathbb{R}$ to be a sequence $(I_0, I_1, \dots)$ of closed rational intervals $[a, b]$ $(a < b, a, b \in \mathbb{Q})$ such that $I_{n+1} \subseteq I_n$ for all $n \in \mathbb{N}$ and:

$$\bigcap_{n \in \mathbb{N}} I_n = \{x\}.$$



We assume, tacitly, that intervals are encoded appropriately, such that, strictly speaking, a name of a real number is an infinite sequence of symbols.

(1.10)

Computable Real Numbers

We call a real number computable iff it has a computable name.

Eg 1 Every rational number $r \in \mathbb{Q}$ is computable.

Proof: Define $I_n = [r - 2^{-n}, r + 2^{-n}]$ for all $n \in \mathbb{N}$. Then the sequence:

$$(I_0, I_1, I_2, \ldots)$$

is a computable name of r. $\square$

Eg 2 $\sqrt{2}$ is computable.

Proof: Define $f : \mathbb{N} \longrightarrow \mathbb{N}$ by $f(n) := \min\{k \in \mathbb{N} \mid k^2 \leq 2n^2 \leq (k+1)^2\}$, $J_0 := [1, 2]$, $J_n := [f(n)/n, (f(n)+2)/n]$, $n > 0$.

Then, the sequence $(J_0, J_1, \ldots)$ is computable, $\sqrt{2} \in J_n$ for all n, and $\lim_{n \to \infty} \text{length}(J_n) = 0$. However, the sequence is not nested in general. We can fix this by defining $I_n := J_0 \cap J_1 \cap \cdots \cap J_n$. Then $(I_0, I_1, \ldots)$ is a computable name of $\sqrt{2}$. $\square$

Eg 3 $\log_3(5)$ is computable.

Proof: $f(n) := \min\{k \in \mathbb{N} \mid 3^k < 5^n < 3^{k+1}\} \ldots$

Computable Real Functions

We call a real function $f :\subseteq \mathbb{R} \to \mathbb{R}$ computable iff some machine maps any name of $x \in dom(f)$ to a name of $f(x)$. For real functions $f :\subseteq \mathbb{R}^n \to \mathbb{R}$, we consider machines reading n names in parallel.

Notice that the machine must behave correctly only for every name of $x \in dom(f)$. For other sequences, it may behave arbitrarily.

It turns out that the elementary real functions exp, sin, log, arcsin ... are computable, and that computable real functions map computable numbers to computable numbers. Moreover, computable real functions are closed under composition.

Eg 3 Real multiplication $(x, y) \mapsto x \cdot y$ is computable.

Proof: For closed $\mathbb{Q}$-intervals I, J, define $I \cdot J = \{x \cdot y \mid x \in I, y \in J\}$. There is a machine with two input streams which maps inputs $(I_0, I_1, \dots), (J_0, J_1, \dots)$ to $(I_0 \cdot I_0, I_1 J_1, \dots)$. If the first is a name of x, the second of y, then clearly the output is a name of $x \cdot y$. □

Example 4

The square root $\sqrt{\cdot} :\subseteq \mathbb{R} \to \mathbb{R}$ is computable.

Proof: Since, for example, $\sqrt{[2:3]}$ is not a rational interval, we must modify the proof of Example 3 for it to work here.

We map each input interval I to a rational interval which is slightly longer than $\sqrt{I}$. For any $a, b \in \mathbb{Q}$ s.t. $0 \leq a < b$, there are rational numbers $r, s \geq 0$ s.t.:

$$a-(b-a) < r^2 \leq a < b \leq s^2 < b+(b-a).$$

Therefore there is a computable function f which for each rational interval I, with no negative elements, determines a rational interval $K = f(I)$ such that $I \subseteq K^2$ and $\text{length}(K^2) < 3 \cdot \text{length}(I)$.

For any rational interval $[a,b]$ with $b \geq 0$, let $g[a,b] := [\max(0,a), b]$. If $(I_0, I_1, \ldots)$ is a name of $x \geq 0$, then $((f\circ g)[I_0], (f\circ g)[I_1], \ldots)$ is a sequence of rational intervals that converges to $\sqrt{x}$. The sequence is not necessarily nested, but we can fix this just as in Example 3. □

Theorem (Continuity)

Every computable real function is continuous!

Proof sketch: We look only at the case $f : \mathbb{R} \to \mathbb{R}$.

Let M be a machine computing f, and let $x \in \mathbb{R}$. We will show f is continuous at x, and thus is continuous, since the choice of x was arbitrary. That is, we will show that $\forall V \in \mathcal{T}_{\mathbb{R}},\ f(x) \in V \Rightarrow \exists U \in \mathcal{T}_{\mathbb{R}},\ x \in U \wedge f[U] \subseteq V$.

So, let V be an open set with $f(x) \in V$. The real number x has a name $([a_0, b_0], [a_1, b_1], \dots)$ such that $a_{i+1} < a_i < b_{i+1} < b_i$. Let $(J_0, J_1, \dots)$ be the name that M produces on this input. Then $J_n \subseteq V$ for some $n \in \mathbb{N}$. Suppose M took k steps to produce the output sequence up to J_n. Then, M can certainly only have read at most k input intervals.

Simply choose $U := (a_k, b_k)$. Clearly $x \in U$. Suppose $y \in U$. Then y has a name of the form $([a_0, b_0], [a_1, b_1], \dots, [a_k, b_k], L_{k+1}, L_{k+2}, \dots)$. On this input, the machine M writes also $(J_0, \dots, J_n)$ within the first k steps, which is the initial part of the name of $f(y)$. Thus $f(y) \in J_n \subseteq V$. Thus $f[U] \subseteq J_n \subseteq V$ as required.

So f is continuous. □

Important Remarks

In our proof of the last theorem, we have used the essential observation that any finite portion of the output of a computation is determined already by a finite portion of its input. However, we did not use that the transformation from inputs to outputs is computable.

As a consequence of this theorem, simple real functions like the step function:

$$s(x) = \begin{cases} 1 & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

and the Gauss staircase $g(x) = \lfloor x \rfloor$ are <u>not</u> computable! These functions are easily definable in our mathematical language, but this does not necessarily imply computability.

As far as we know, the step function, or indeed any non-continuous function, cannot be computed by physical devices.

N.B. One of the reasons there is no agreement on the foundations of computable analysis is the issue that this definition does not say that such simple functions are computable.

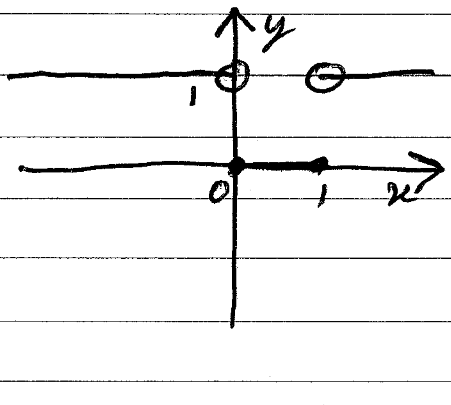
Subsets of $\mathbb{R}$: Recursive

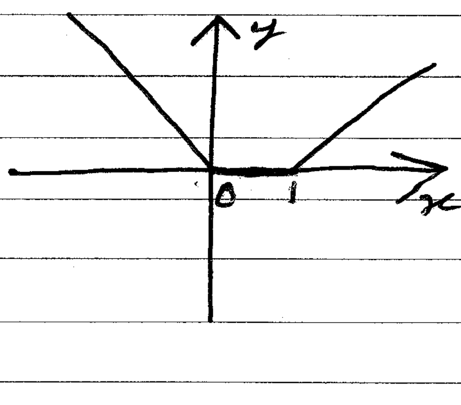
A subset $A \subseteq \mathbb{N}$ is recursive (decidable) iff its characteristic function $cf_A : \mathbb{N} \to \mathbb{N}$ is computable. Suppose we extended this definition to $\mathbb{R}$. Then, by the continuity theorem, the only computable subsets would be $\emptyset$ and $\mathbb{R}$, so that definition is useless.

The distance function $d_A : \mathbb{R}^n \to \mathbb{R}$ defined by $d_A(x) := \inf_{y \in A} |y - x|$ is a more useful generalisation of the discrete characterisation function. We call a (closed) subset $A \subseteq \mathbb{R}^n$ recursive iff its distance function $d_A : \mathbb{R}^n \to \mathbb{R}$ is computable.

It turns out that closed intervals $[a,b]$ with computable endpoints, and the graph of any computable $f : \mathbb{R} \to \mathbb{R}$ are recursive.

Eg 5 Compare the graphs of $1 - cf_{[0,1]}(x)$ and $d_{[0,1]}(x)$:





Subsets of $\mathbb{R}$: Computability

For defining computability on subsets of $\mathbb{R}$, we introduce naming systems of sets of subsets. Unfortunately $2^{\mathbb{R}}$ is too large, so we must restrict ourselves. For now, we will only consider the open subsets of $\mathbb{R}$: $\mathcal{O}(\mathbb{R})$ $(= \tau_{\mathbb{R}})$.

We define a name of an open subset $U \subseteq \mathbb{R}$ to be a sequence $(I_0, I_1, \dots)$ of open intervals with rational boundaries such that $U = I_0 \cup I_1 \cup I_2 \cup \cdots$. In the general case, $U \subseteq \mathbb{R}^n$, each I_k is a product of n open intervals with rational boundaries. We call an open set recursively enumerable iff it has a computable name. As for real functions, we call a function on $\mathcal{O}(\mathbb{R})$ computable iff some machine maps names of arguments to names of results.

Eg 6 The open intervals (a, b) with computable endpoints are recursively enumerable. So is the set $\{(x,y) \in \mathbb{R}^2 \mid x < y\}$. In fact $\{(x,y) \subseteq \mathbb{R}^2 \mid x \leq y\}$ is recursive.

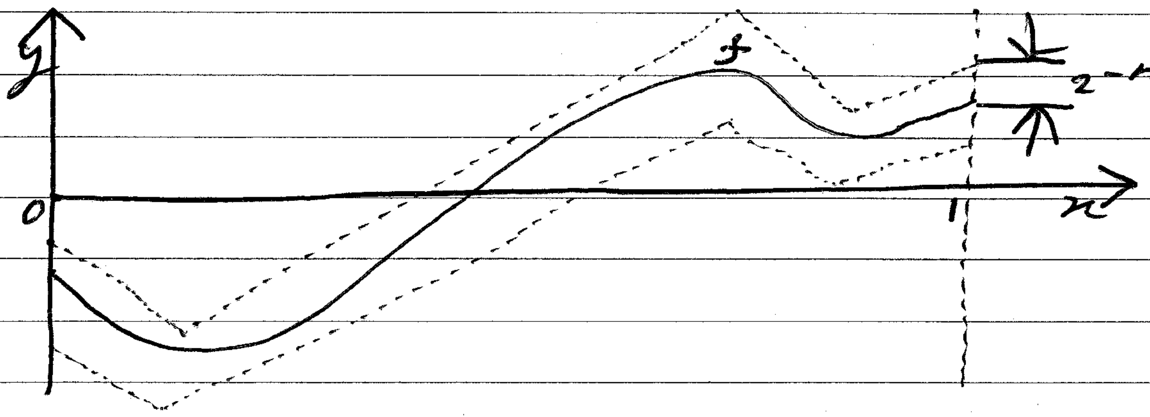
Eg 7 The functions intersection and union on open sets are computable.

The Space $C[0,1]$

We call $f:[0,1]\to\mathbb{R}$ a rational polygon iff its graph is a polygon with finitely many vertices, with $\mathbb{Q}$-coordinates. Define the function ball with center $p\in C[0,1]$, radius $r>0$: $B(p,r):=\{f\in C[0,1] \mid d(f,p)<r\}$ where $d(f,p)=\max_{x\in[0,1]}|f(x)-p(x)|$.

A name of a continuous function $f:[0,1]\to\mathbb{R}$ is a sequence $(B_0, B_1, \ldots)$ where B_n is a function ball of radius 2^{-n} with $f\in B_n$ the center of which is a rational polygon.

Eg 8



A name $(B_0, B_1, \ldots)$ of f encloses f arbitrarily narrowly. A function $f:[0,1]\to\mathbb{R}$ can have a computable name, and it can be computable (by a machine transforming each name of the real numbers $x\in[0,1]$ to a name of $f(x)$). It will turn out these notions are equivalent!

Closing Remarks

Our naming systems of $\mathbb{R}$ and $C[0,1]$ make the evaluation function:

$$Apply: C[0,1] \times [0,1] \rightarrow \mathbb{R}$$

defined by $Apply(f, x) := f(x)$, computable. That is, there is a machine which transforms any name of any f and any name of any x to a name of $f(x)$.

There are many other naming systems of $C[0,1]$ for which the evaluation function becomes computable, however, among all of these, this is the "weakest".

Eg 9 Integration $f \mapsto \int_0^1 f(x)\,dx$ is an important computable operator (when $f \in C[0,1]$), while differentiation $f \mapsto f'$ for continously differentiable $f \in C[0,1]$ is not computable.

Eg 10 By the IVT, every $f \in C[0,1]$ with $f(0) < 0 < f(1)$ has a zero. Unfortunately, there is no computable operator for zero-finding working correctly for all $f \in C[0,1]$ with $f(0) < 0 < f(1)$.

However, if we require that f is also increasing, then the operator that returns the unique zero is computable.

Appendix: Full Quote of L. Blum '96

Our perspective is to formulate the laws of computation. Thus, we write not from the point of view of the engineer who looks for a good algorithm which solves the problem at hand, or wishes to design a faster computer. The perspective is more like that of a physicist, trying to understand the laws of scientific computation. [...]

There is a substantial conflict between theoretical computer science and numerical analysis. These two subjects with common goals have grown apart. For example, computer scientists are uneasy with calculus, while numerical analysts thrive on it. On the other hand, numerical analysts see no use for the Turing machine.

The conflict has its roots in ~~the~~ another age-old conflict, that between the continuous and the discrete. [...] Algorithms are primarily a means to solve practical problems. There is not even a formal definition of algorithm in the subject [of numerical analysis]. [...] Thus, we view numerical analysis as an eclectic subject with weak foundations; this certainly in no way denies the great achievements through the centuries.

Recap

Classically, computability is introduced for functions $f :\subseteq (\Sigma_1^*)^n \longrightarrow \Sigma_1^*$ 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.

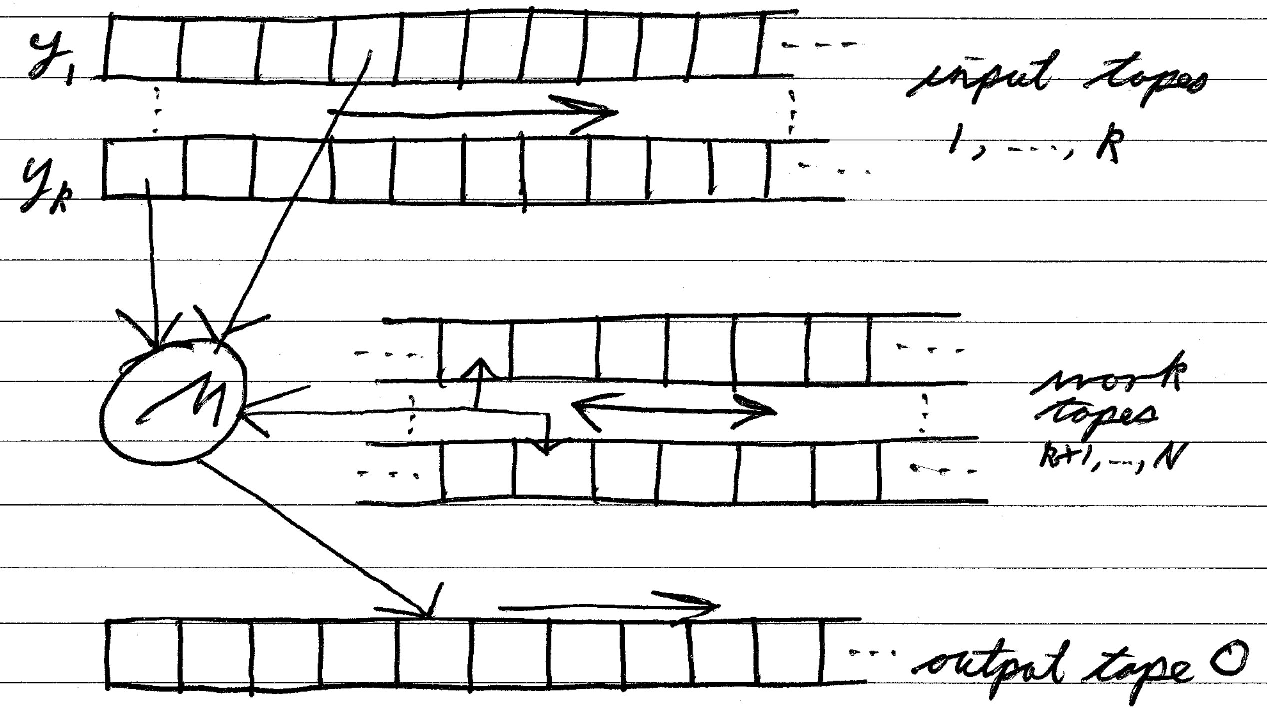
Equivalently, one can start with computable functions on the natural numbers, instead of words, and use numbers as names. Since Σ_1^* is only countable, this method cannot be applied for introducing computability on uncountable sets M like $\mathbb{R}$, $\mathcal{O}(\mathbb{R})$, $2^{\mathbb{R}}$, $C[0,1]$.

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_1 with a special symbol $B \in \Sigma_1$, a work alphabet T^1 s.t. $\Sigma_1 \cup \{B\} \subseteq T^1$, a number k of input tapes, and a "flowchart" (relation) operating on the tapes.

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

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 $Y_i \in \{\Sigma_1^{*}, \Sigma_1^{\omega}\}$, 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 $Y_0 = \Sigma_1^{*}$: $f_M(y_1, \ldots, y_k) := y_0 \in \Sigma_1^{*}$ 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 $f_M(y_1, \ldots, y_k)$ 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^{*} and Σ_1^{ω} straightforwardly:

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

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

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

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

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

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

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

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

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 ~~by~~ be a Type-2 machine computing $f :\subseteq \Sigma_1^{\omega} \to \Sigma_1^{*}$. Define $h : \Sigma_1^{*} \to \Sigma_1^{*}$ by

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

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 $p_{<i}$ in t steps, printing $h(p_{<i})$. Since $dom(h)$ is prefix-free, there is at most one such k, so N computes the function h_*. $\square$

Characterization of Σ_1^ω

<u>Dfn</u> For any monotone (w.r.t. $\sqsubseteq$) total function $h: \Sigma_1^{*} \to \Sigma_1^{*}$, define $h_\omega : \subseteq \Sigma_1^\omega \to \Sigma_1^\omega$ by:

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

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

<u>Lem.</u> 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 $f = h_\omega$. $\square$

Closure under Composition

Thm For $k, n \in \mathbb{N}$ and $X_1, \dots, X_k, Y_1, \dots, Y_n, Z \in \{\Sigma_1^*, \Sigma_1^\omega\}$, let $g_i :\subseteq X_1 \times \dots \times X_k \to Y_i$, $f :\subseteq Y_1 \times \dots \times Y_n \to Z$ be computable functions. Then the composition

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

is computable if $Z = \Sigma_1^\omega$ or $Y_i = \Sigma_1^*$ for all i.

Alternatively, if $Z = \Sigma_1^*$ and $Y_i = \Sigma_1^\omega$ for some i, then it has a computable extension:

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

with $dom(h) \cap dom(g_1, \dots, g_n) = dom(f \circ (g_1, \dots, g_n))$.

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 $f : \{()\} \to \Sigma_1^\omega ; () \mapsto p$ is. Then just apply the above theorem. $\square$

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^{*} and Σ_1^{ω}

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

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

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

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

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

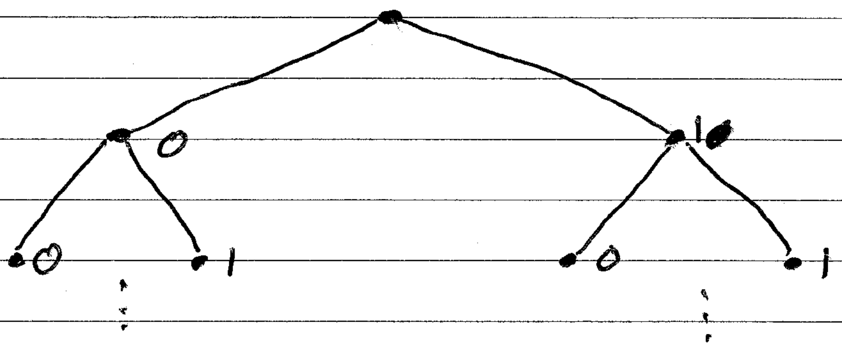
Finally, since $A\Sigma_1^{\omega} = \bigcup\{w\Sigma_1^{\omega} \mid w \in A\}$ for each $A \subseteq \Sigma_1^*$, β must be a base of T_C.

Product topologies can be defined with the canonical base $\beta_Y := \{y \circ Y \mid y \in (\Sigma^*)^*\}$ where $Y = Y_1 \times \cdots \times Y_k$ $(Y_1, \ldots, Y_k \in \{\Sigma_1^*, \Sigma_1^{\omega}\})$ and $(y_1, \ldots, y_k) \circ Y := U_1 \times \cdots \times U_k$ with:

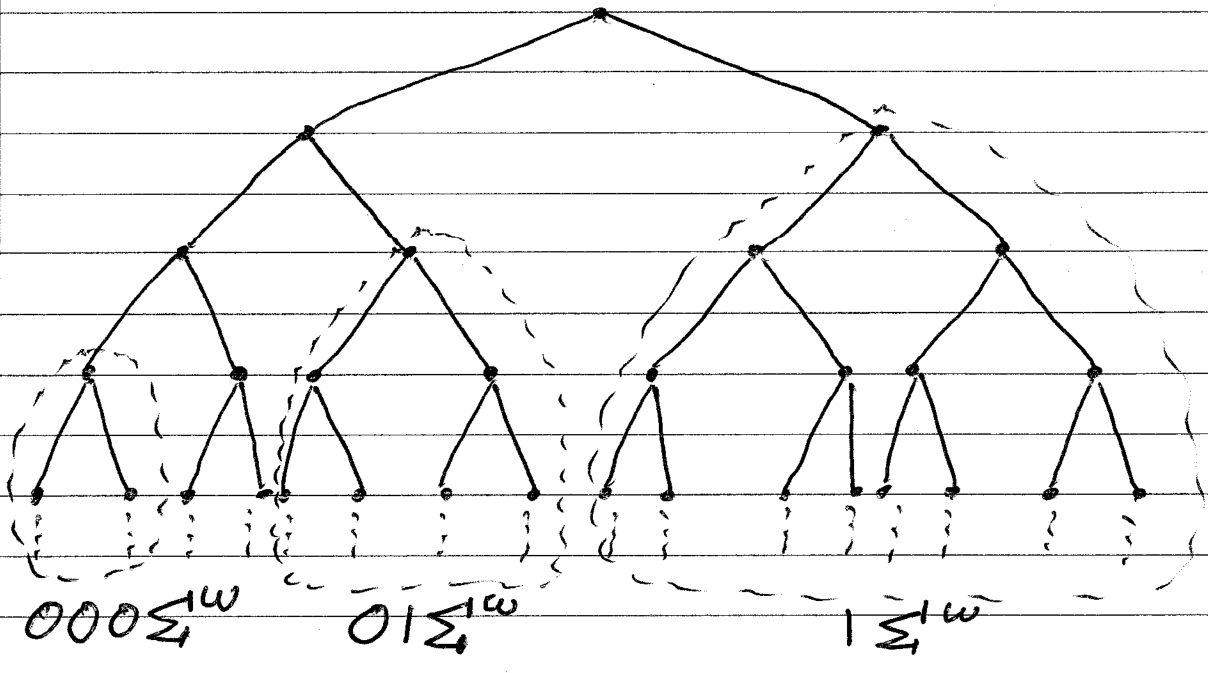
$$U_i := \begin{cases} \{y_i\} & \text{if } Y_i = \Sigma_1^* \\ y_i\Sigma_1^{\omega} & \text{if } Y_i = \Sigma_1^{\omega}. \end{cases}$$

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 $w \in \Sigma_1^{\omega}$ with $p \in w\Sigma_1^{\omega} \subseteq f_M^{-1}[u\Sigma_1^{\omega}]$. That is, f_M is (τ_c, τ_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. $\square$

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 $(y_1, \ldots, y_k) \in \mathrm{dom}(f)$ has an open neighbourhood $(u_1, \ldots, u_k) \circ Y \subseteq \mathrm{dom}(f)$. 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), $\mathrm{dom}(f_{M_n})$ is open for each n, so $\mathrm{dom}(f_M) = \bigcap \mathrm{dom}(f_{M_n})$ 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 comactness, 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 $(u_1, v_1), \ldots, (u_k, v_k) \in \Sigma_1^* \times \Sigma_1^*$ such that :

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

(2) $\Sigma_1^\omega = u_1\Sigma_1^\omega \cup \cdots \cup u_k\Sigma_1^\omega$

(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^ω, 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.

Standard Representations

In mathematical logic, a Gödel numbering is a function that assigns each symbol and well-formed formula to a unique natural number, used by K. Gödel for the proof of his incompleteness theorems in 1931.

Equivalently, a Gödel numbering $\varphi : \mathbb{N} \to P^{(1)}$ of the set $P^{(1)}$ of partial recursive functions (computable number functions) $f : \subseteq \mathbb{N} \to \mathbb{N}$ is defined uniquely up to equivalence in Type-1 recursion theory. This is due to the utm-theorem and smn-theorem.

The utm-theorem (Universal Turing Machine Theorem) affirms the existence of a computable universal function which is capable of calculating any other computable function, an abstraction of the Universal Turing Machine.

In practical terms, the smt-theorem (Translation Lemma or Parameter Theorem) says that for a given programming language and $m, n \in \mathbb{N}$, there exists a particular algorithm that accepts as input the source code of a program with $m+n$ free variables together with m values. This algorithm generates source code that substitutes the values for the first m free variables, leaving the rest free.

Notation and Representation

In order to produce Type-2 generalizations of these theorems, we must first introduce general notations and representations. A "notation" of a set M is a surjective function $\nu :\subseteq \Sigma_1^{*} \rightarrow M$, and a "representation" is a surjective function $\delta :\subseteq \Sigma_1^{\omega} \rightarrow M$. A "naming system" is a notation or a representation.

Sometimes we will say that $p \in Y$ is a γ-name of $x \in M$ if $\gamma :\subseteq Y \rightarrow M$ is a naming system and $\gamma(p) = x$. Notice that we do not consider functions $\gamma :\subseteq \Sigma_1^{*} \cup \Sigma_1^{\omega} \rightarrow M$ as names.

For $\gamma :\subseteq Y \rightarrow M$, $\gamma' :\subseteq Y' \rightarrow M'$ with $Y, Y' \in \{\Sigma_1^{*}, \Sigma_1^{\omega}\}$, we define:

(1) $f :\subseteq Y \rightarrow Y'$ reduces γ to γ' iff $\gamma(y) = (\gamma' f)(y)$ for all $y \in dom(\gamma)$.

We write $\gamma \leq \gamma'$ if f is computable and $\gamma \leq_t \gamma'$ if f is continuous. $\leq$ and $\leq_t$ are preorders on the class of naming systems.

(2) $\gamma \equiv \gamma'$ iff $\gamma \leq \gamma' \leq \gamma$, and $\gamma \equiv_t \gamma'$ iff $\gamma \leq_t \gamma' \leq_t \gamma$. $\equiv$ and $\equiv_t$ are equivalences on the class of naming systems.

utm/smn - Properties Revisited

In elementary computability theory, Turing machines computing word functions $f :\subseteq \Sigma_1^* \to \Sigma_1^*$ are encoded canonically by words from Σ_1^*. If $\psi_w :\subseteq \Sigma_1^* \to \Sigma_1^*$ is the word function computed by the Turing machine with code w, then the partial function (partial notation) $w \mapsto \psi_w$ of the computable word functions satisfies the utm and smn-properties:

utm(ψ): The function $(w, x) \mapsto \psi_w(x)$ is computable.

smn(ψ): For every computable function $f :\subseteq \Sigma_1^* \times \Sigma_1^* \to \Sigma_1^*$, there is a total computable function $r : \Sigma_1^* \to \Sigma_1^*$ with $r(x) \in \mathrm{dom}(\psi)$ for all $x \in \Sigma_1^*$ and $f(x, y) = \psi_{r(x)}(y)$ for all $x, y \in \Sigma_1^*$.

Thm utm(ψ) and smn(ψ) hold.

Proof: See any standard text for detail, including for detail of universal encoding. $\square$

General utm/smn-Properties

Let $a, b, c \in \{*, \omega\}$, G^{ab} be a set of functions $g :\subseteq \Sigma_1^a \to \Sigma_1^b$, $\xi :\subseteq \Sigma_1^c \to G^{ab}$ be a naming system of G^{ab}. Define:

utm(ξ): There is a computable (universal) function $u :\subseteq \Sigma_1^c \times \Sigma_1^a \to \Sigma_1^b$ with $\xi_x(y) = u(x, y)$ for all $x \in \mathrm{dom}(\xi)$ and $y \in \Sigma_1^a$.

smn(ξ): For every computable function $f :\subseteq \Sigma_1^c \times \Sigma_1^a \to \Sigma_1^b$ there is a total computable function $s : \Sigma_1^c \to \Sigma_1^c$ such that $s(x) \in \mathrm{dom}(\xi)$ for all $x \in \Sigma_1^c$ and $f(x, y) = \xi_{s(x)}(y)$ for all $x \in \Sigma_1^c$ and $y \in \Sigma_1^a$.

Consider a canonical encoding of Type-2 Machines with one input tape by words $w \in \Sigma_1^*$ such that the set TC of code words is recursive. For all $a, b \in \{*, \omega\}$, define:

$$P^{ab} := \{f :\subseteq \Sigma_1^a \to \Sigma_1^b \mid f \text{ is computable}\}$$

and a notation $\xi^{ab} : \Sigma_1^* \to P^{ab}$ of the set P^{ab} defined by $\xi^{ab}(w)$ is undefined on non-code words, otherwise the function $f \in P^{ab}$ computed by the Type-2 machine with code w.

<u>Thm</u> For all $a, b \in \{*, \omega\}$, we have utm(ξ^{ab}) and smn(ξ^{ab}).

Classes of Continuous Functions

As well as representations of functions, we also would like representations of sets of functions. Obviously, we can only consider sets of size at most the continuum cardinality, as we have discussed.

Define:

$$F^{**} := \{f \mid f :\subseteq \Sigma_1^* \to \Sigma_1^*\},$$
$$F^{*\omega} := \{f \mid f :\subseteq \Sigma_1^* \to \Sigma_1^\omega\},$$
$$F^{\omega *} := \{f \mid f :\subseteq \Sigma_1^\omega \to \Sigma_1^* \text{ continuous} \wedge \operatorname{dom}(f) \text{ open}\},$$
$$F^{\omega\omega} := \{f \mid f :\subseteq \Sigma_1^\omega \to \Sigma_1^\omega \text{ continuous} \wedge \operatorname{dom}(f)\ G_\delta\}.$$

Thm $F^{\omega *} = \{h_* \mid h :\subseteq \Sigma_1^* \to \Sigma_1^* \text{ has prefix-free domain}\}$
$F^{\omega\omega} = \{h_\omega \mid h : \Sigma_1^* \to \Sigma_1^* \text{ is monotone}\}$

By the next theorem, $F^{\omega *}$, $F^{\omega\omega}$ represent essentially all partial and continuous functions $f :\subseteq \Sigma_1^\omega \to \Sigma_1^*$ and $f :\subseteq \Sigma_1^\omega \to \Sigma_1^\omega$, respectively.

Thm Every ctns $f :\subseteq \Sigma_1^\omega \to \Sigma_1^*$ has an extension in $F^{\omega *}$ and every ctns $f :\subseteq \Sigma_1^\omega \to \Sigma_1^\omega$ has an extension in $F^{\omega\omega}$.

Finally, functions from F^{ab} are essentially closed under composition for $a, b \in \{*, \omega\}$.

Thm If $g \in F^{ab}$, $f \in F^{bc}$ where $a, b, c \in \{*, \omega\}$. If $b = \omega$ and $c = *$ then $f \circ g$ has an extension $d \in F^{a*}$ with $\operatorname{dom}(d) \cap \operatorname{dom}(g) = \operatorname{dom}(f \circ g)$. Otherwise, $f \circ g \in F^{ac}$.

Standard Representations of F^{ab}

For all $a, b \in \{*, \omega\}$, define the standard rep. $\eta^{ab} : \Sigma_1^{\omega} \longrightarrow F^{ab}$ of F^{ab} by:

$$\eta^{ab}(\langle x, p\rangle)(y) := \xi_x^{\omega b}\langle p, y\rangle$$

for all $x \in \Sigma_1^{*}$, $p \in \Sigma_1^{\omega}$, $y \in \Sigma_1^{a}$, and

$$\eta^{ab}(q)(\_) := \bot$$

if for no $x \in \Sigma_1^{*}$, $\iota(x)$ is a prefix of q.

Therefore, roughly speaking, $\eta^{ab}_{\langle x, q\rangle}(y)$ is the result of the Type-2 machine M with code x applied to input.

Lem Representations η^{ab} are well-defined.

Lem Computable functions have computable names. That is, a function $f : \subseteq \Sigma_1^{a} \longrightarrow \Sigma_1^{b}$ is computable iff $f = \eta^{ab}_p$ for some computable $p \in \Sigma_1^{\omega}$.

Thm For each $a, b \in \{*, \omega\}$, we have $utm(\eta^{ab})$ and $smn(\eta^{ab})$.

An Equivalence Theorem

In 1967, H. Rogers proved that for the numbering $\varphi : \mathbb{N} \longrightarrow P^{(1)}$, up to equivalence, is the only numbering of $P^{(1)}$ satisfying the utm-theorem and smn-theorem. This can be easily generalised to our naming systems of function spaces.

Thm For notations β, γ, δ of G^{ab} with $utm(\delta)$ and $smn(\delta)$, we have:

$$(1)\ (\beta \leq \gamma \wedge utm(\gamma)) \Rightarrow utm(\beta),$$
$$(2)\ (smn(\beta) \wedge \beta \leq \gamma) \Rightarrow smn(\gamma),$$
$$(3)\ (utm(\beta) \wedge smn(\gamma)) \Rightarrow \beta \leq \gamma,$$
$$(4)\ utm(\beta) \Leftrightarrow \beta \leq \delta,$$
$$(5)\ smn(\beta) \Leftrightarrow \delta \leq \beta,$$
$$(6)\ (utm(\gamma) \wedge smn(\gamma)) \Leftrightarrow \gamma \equiv \delta.$$

Finally, we can think about this theorem and the last, purely topologically.

Thm We have $tutm(\eta^{ab})$ and $tutm(\eta^{ab})$, where, for a representation of F^{ab}, δ:

$tutm(\delta)$: There is a $u \in F^{\omega b}$ st. $u\langle p, y\rangle = \delta_p(y)$ for all $p \in dom(\delta)$ and $y \in \Sigma_1^{a}$.

$tsmn(\delta)$: For every $f \in F^{\omega b}$, there is a cts $S : \Sigma_1^{\omega} \longrightarrow \Sigma_1^{\omega}$ with $S(p) \in dom(\delta)$ and $f\langle p, y\rangle = \delta_{S(p)}(y)$ for all $p \in \Sigma_1^{\omega}$ and $y \in \Sigma_1^{a}$.

Decidable Sets

Recall from Type-1 theory that a subset $A \subseteq \Sigma_1^*$ is called recursive or decidable iff its characteristic function is computable, and is called recursively enumerable (r.e.) iff it is the domain of a computable function $f :\subseteq \Sigma_1^* \to \Sigma_1^*$. A is recursive iff A and A^c are both r.e.

We will generalise these concepts for Type-2 theory, opting for the terminology of "decidable", ~~Reg~~ defining "recursive" later.

Consider $X \subseteq Z \subseteq Y := Y_1 \times \cdots \times Y_k$ where $k \geq 1$ and $Y_1, \ldots, Y_k \in \{\Sigma_1^*, \Sigma_1^\omega\}$. Then:

(1) X is called r.e. open in Z iff $X = dom(f) \cap Z$ for some computable function $f :\subseteq Y \to \Sigma_1^*$.

(2) X is called decidable in Z iff both X and $Z \setminus X$ are r.e. open in Z.

(3) X is open in Z iff $X = U \cap Z$ for some open set $U \subseteq Y$, and closed in Z iff $Z \setminus X$ is open in Z.

Observe that X r.e. open (decidable) in Z $\Rightarrow$ X is open (clopen) in Z. This follows from our theorem from 2.15.

A Characterization Theorem

Thm Let $X \subseteq Z \subseteq Y$ be as before. Then X is clopen (decidable) in Z iff there is a continuous (computable) function $f :\subseteq Y \to \Sigma_1^*$ with $f(z) = 1$ if $z \in X$ and $f(z) = 0$ if $z \in Z \setminus X$.

$\Rightarrow$ Proof 1: Let X and $Z \setminus X$ be open in Z. Then there are open sets U, V such that $X = U \cap Z$ and $Z \setminus X = V \cap Z$. Define $f :\subseteq Y \to \Sigma_1^*$ by $\mathrm{dom}(f) := Z$, $f(z) := 1$ if $z \in X$, $f(z) := 0$ if $z \in Z \setminus X$. Then clearly f meets the requirements and is continuous since $f^{-1}[\{1\}] = U \cap \mathrm{dom}(f)$, $f^{-1}[\{0\}] = V \cap \mathrm{dom}(f)$.

$\Leftarrow$ Let f be as above. Then $f^{-1}[\{1\}] = U \cap \mathrm{dom}(f)$ for some open set U. So $X = f^{-1}[\{1\}] \cap Z = U \cap Z$, thus X is open in Z. Similarly, $Z \setminus X$ is open in Z. $\square$

$\Rightarrow$ Proof 2: Let X and $Z \setminus X$ be r.e. open in Z. Then there are Type-2 machines M and N with $\mathrm{dom}(f_M) \cap Z = X$ and $\mathrm{dom}(f_N) \cap Z = Z \setminus X$. Clearly we can construct a machine L that runs M and N in parallel, halting as soon as either halts, writing either 1 or 0 if M or N halts first, respectively. Set $f := f_L$.

$\Leftarrow$ Let L be the machine that computes f, as above. From L, we construct M which on input $z \in Y$ halts iff L halts on z with result 1. Then $X = \mathrm{dom}(f_M) \cap Z$, that is, X is r.e. open in Z. Similarly, $Z \setminus X$ is r.e. open in Z. $\square$

Some Examples

Recall that each decidable set $X \subseteq \Sigma_1^\omega$ has the form $X = A\Sigma_1^\omega$ for some finite $A \subseteq \Sigma_1^*$.

Eg 1 Clearly Z is decidable in Z.

Eg 2 If X is r.e. open (decidable), then X is r.e. open (decidable) in Z whenever $X \subseteq Z$.

Eg 3 The set $X = \{p \in \Sigma_1^\omega \mid p \neq 0^\omega\}$ is r.e. open, but its complement is not, since it does not contain a set $w\Sigma_1^\omega$. Therefore, X is not closed, so not clopen, so not decidable.

Eg 4 $X \subseteq Y_1 \times \cdots \times Y_k$ is r.e. open (decidable) iff $\langle X \rangle$ is r.e. open (decidable) in $\langle Y_1 \times \cdots \times Y_k \rangle$.

Eg 5 Union and intersection of r.e. open sets are r.e open. For decidable sets, complement is decidable too.

Eg 6 If $f :\subseteq Y_1 \times \cdots \times Y_k \to Y_0$ is computable and $U \subseteq W \subseteq Y_0$ is r.e. open (decidable) in W, then $f^{-1}[U]$ is r.e. open (decidable) in $f^{-1}[W]$.

Another Characterization Theorem

Thm For $X \subseteq Y := Y_1 \times \cdots \times Y_k$ ($Y_1, \ldots, Y_k \in \{\Sigma_1^{*}, \Sigma_1^{\omega}\}$), the following properties are equivalent:

(1) X is r.e. open
(2) $X = A \circ Y$ for some r.e. $A \subseteq (\Sigma_1^{*})^k$
(3) X is open and $\{y \in (\Sigma_1^{*})^k \mid y \circ Y \subseteq X\}$ is r.e.

Recursive Open Sets

For any open $X \subseteq Y := Y_1 \times \cdots \times Y_k$, define:

X is r.e. $:\Leftrightarrow \{y \in (\Sigma_1^{*})^k \mid y \circ Y \subseteq X\}$ is r.e.

X is co-r.e. $:\Leftrightarrow \{y \in (\Sigma_1^{*})^k \mid y \circ Y \not\subseteq X\}$ is r.e.

X is recursive $:\Leftrightarrow \{y \in (\Sigma_1^{*})^k \mid y \circ Y \subseteq X\}$ is decidable.

~~Thus~~ Thus, an open set $X \subseteq Y$ is recursive iff it is both r.e. and co-r.e. Notice that if Y is $(\Sigma_1^{*})^k$, then X is r.e. iff it is r.e. in the usual sense, and similarly for recursive.

Recall that the r.e. open subsets of Y are closed under union and intersection. The recursive open sets are not closed under union, but are under intersection. Decidable $\Rightarrow$ Recursive, but the reverse is not necessarily true.

Taking Stock

So far, we have defined computability on the sets Σ_i^* of finite words and Σ_i^ω of infinite sequences by Type-2 machines. In TTE we induce computability on other sets M by using finite or infinite words as "names". Machines, therefore, still transform "concrete" sequences of symbols. We have seen that some concepts of computability theory have similar topological counterparts.

In particular, we are interested in computability concepts induced by naming systems. Our next ~~seteps~~ must be to transfer our notions for Σ_i^* and Σ_i^ω to sets M by means of naming systems. We will then introduce and discuss the important class of "admissible" representations, at which point we will then be able to return to our original motivating questions.

For means of simplicity, we do not consider naming systems where both finite and infinite symbol sequences are used as names simultaneously. If $\gamma :\subseteq Y \to M$ is a naming system, then every element of M is named since γ is surjective, but not every $y \in Y$ is a name, necessarily. Furthermore, an element of M may have several names. We have informally seen this for the reals, represented by decreasing rational intervals.

Summary of Concepts

From Type-1 theory, we have Turing Machines, partial (computable) functions, numberings, and recursive and r.e. sets $A \subseteq \Sigma_1^*$.

From Topology, we have continuity and compactness.

From Type-2 theory, we have:

(1) Type-2 Machines
(2) (Computable) String Functions
(3) (Computable) Elements
(4) Tupling Functions
(5) Essentially closure under composition
(6) Primative recursion
(7) The finiteness property
(8) Discrete and Cantor topologies
(9) Computable $\Rightarrow$ Continuous
(10) Naming and reduction
(11) UTM and smn-theorems
(12) Classes of (continuous) functions
(13) (Continuous) extension
(14) r.e. open and decidable
(15) co-r.e. open and recursive open

3.14

Appendix: Proof of Theorem

For simplicity we consider only case $Y = \Sigma_1^{*} \times \Sigma_1^{\omega}$.

1 ⇒ 2 Let X be r.e. open. Then $X = dom(f_M)$ for some M. Define a computable function $g :\subseteq \Sigma_1^{*} \times \Sigma_1^{*} \to \Sigma_1^{*}$ by:

$$g(x_1, x_2) := \begin{cases} 1 & \text{if } M \text{ halts on input } (x_1, x_2 0^\omega) \text{ after reading at most the first } |x_2| \text{ symbols from the second input tape} \\ \perp & \text{otherwise} \end{cases}$$

$A := g^{-1}[\{1\}]$ is r.e.. If $(x_1, x_2) \in A$, then M halts on input $(x_1, x_2 q)$ for every $q \in \Sigma_1^{\omega}$, so $(x_1, x_2) \circ Y \subseteq X$, thus $A \circ Y \subseteq X$. On the other hand, if $(x_1, p) \in X$, then for some prefix x_2 of p, M halts on input (x_1, p) after reading at most the first $|x_2|$ symbols from the second input tape. Therefore, $(x_1, x_2) \in A$ and $(x, p) \in A \circ Y$, thus $X \subseteq A \circ Y$. □

2 ⇒ 1 Let M be a TM with $A = dom(f_M)$, and N the Type-2 machine which on input $(x, p) \in \Sigma_1^{*} \times \Sigma_1^{\omega}$ searches for the smallest $n = \langle k, m \rangle$ such that M on $(x, p_{<k})$ halts in m steps as soon as it has found n. So $A \circ Y = dom(f_N)$ □

2 ⇒ 3 Suppose $y \circ Y \subseteq A \circ Y$. Since $y \circ Y$ is a compact subset of Y, there is a finite C s.t. $y \circ Y \subseteq C \circ Y$. Since this property is decidable in y and C, if A is r.e. then $\{y \in (\Sigma_1^{*})^k \mid y \circ Y \subseteq A \circ Y\}$ is. Clearly X is open if $X = A \circ Y$. □

3 ⇒ 2 If X is open and $B := \{y \in (\Sigma_1^{*})^k \mid y \circ Y \subseteq X\}$, then $X = B \circ Y$. □