König's Lemma and Kleene Tree

Andrej Bauer

May 3, 2006

Abstract

I present a basic result about Cantor space in the context of computability theory: the computable Cantor space is computably non-compact. This is in sharp contrast with the classical theorem that Cantor space is compact. The note is written for mathematicians with classical training in topology and analysis. I assume nothing from computability theory, except the basic intuition about how computers work by executing instructions given by a finite program.

1 Trees

We first review some basic definitions and facts about binary trees. Let $2 = \{0, 1\}$ be the two-point discrete space. We let 2^* be the set of finite binary sequences,

$$2^* = \{a_1 a_2 \dots a_n \mid n \geq 0, a_i \in 2\} .$$

We denote finite sequences with letters a, b, c, ... The empty sequence is denoted by ε. If $a = a_1 \dots a_n$ is a finite sequence, $|a| = n$ is its *length*. We say that a is a *prefix* of b and write $a \sqsubseteq b$, when $|a| \leq |b|$ and $a_k = b_k$ for $1 \leq k \leq |a|$. The prefix relation $\sqsubseteq$ is a partial order on 2^*.

Let 2^ω be the set of infinite binary sequences, which we denote with Greek letters α, β, γ, ... A finite sequence $a \in 2^*$ is a *prefix* of $\alpha \in 2^\omega$, written $a \sqsubseteq \alpha$, when $a_k = \alpha_k$ for all $1 \leq k \leq |a|$. The *n-th prefix* of α is the finite sequence $\alpha_1 \dots \alpha_n$.

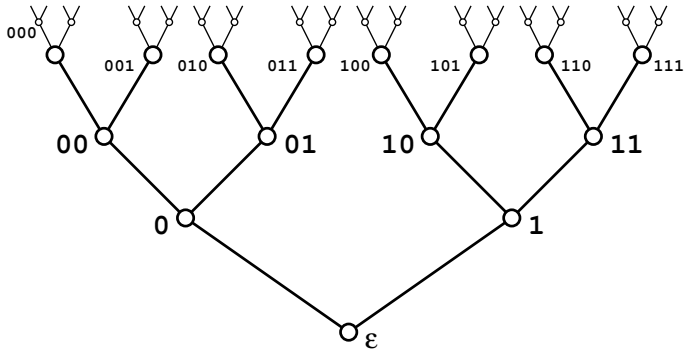

Figure 1: The full infinite binary tree

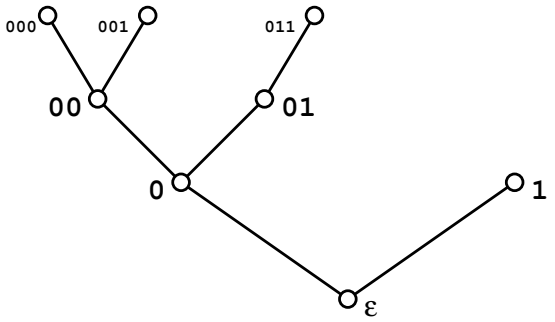

Figure 2: The tree $T = \{\varepsilon, 0, 1, 00, 01, 000, 001, 011\}$

We may picture 2^* as the *full (infinite binary) tree* shown in Figure 1. Each node in the tree corresponds a finite binary sequence and vice versa. In general, a *(binary) tree* is a non-empty prefix-closed subset of 2^*. A subset $T \subseteq 2^*$ is said to be *prefix-closed* if $b \in T$ and $a \sqsubseteq b$ implies $a \in T$. The *leaves* of a tree T are the maximal elements in the $\sqsubseteq$ order. For example, the set $T = \{\varepsilon, 0, 1, 00, 01, 000, 001, 011\}$ is a tree whose picture is shown in Figure 2. The leaves of T are 000, 001, 011, and 1. Note that a tree may be finite or infinite.

A *path* is a tree in which every node has at most one successor with respect to prefix order. A path may be finite or infinite, and is equal to the set of all prefixes of a unique (finite or infinite) sequence. Often we do not distinguish between a sequence and the path of its prefixes.

2 König's Lemma

It is well known that Cantor space, which is 2^ω with the product topology, is compact. Interestingly, compactness can be proved without the Axiom of Choice,[1] and is equivalent[2] to a statement about binary trees, known as Weak König's Lemma. We focus on the lemma rather than on the Heine-Borel property of Cantor space because in computability theory it is easier to handle binary trees than general open covers.

Lemma 2.1 (König) *An infinite binary tree contains an infinite path.*

Proof. Suppose T is an infinite tree. For $a \in 2^*$ define

$$T_a = \{b \in T \mid a \sqsubseteq b\} .$$

The set T_a is that part of T which lies above the node a. We construct an infinite sequence α by induction in such a way that for every n the set $T_{\alpha_1 \ldots \alpha_n}$ is infinite.

When $n = 0$ we have $T_\varepsilon = T$ which takes care of the base case. For the induction step, suppose the first $n \geq 0$ terms of α have been constructed so

[1] In contrast, the general Tychonoff theorem is equivalent to the Axiom of Choice.

[2] We need to be careful what we mean by "equivalence", see discussion at the end of the section.

that $T_{\alpha_1 \ldots \alpha_n}$ is infinite. Because $T_{\alpha_1 \ldots \alpha_n} = T_{\alpha_1 \ldots \alpha_n 0} \cup T_{\alpha_1 \ldots \alpha_n 1}$, at least one of $T_{\alpha_1 \ldots \alpha_n 0}$ or $T_{\alpha_1 \ldots \alpha_n 1}$ must be infinite. Thus if we define

$$\alpha_{n+1} = \begin{cases} 0 & \text{if } T_{\alpha_1 \ldots \alpha_n 0} \text{ is infinite,} \\ 1 & \text{otherwise,} \end{cases}$$

it will be the case that $T_{\alpha_1 \ldots \alpha_{n+1}}$ is infinite. To complete the proof, observe that the prefixes of α form an infinite path in T. ∎

Let us prove that Weak König's Lemma implies compactness of Cantor space. For $a \in 2^*$ the set

$$B_a = \{\alpha \in 2^\omega \mid a \sqsubseteq \alpha\}$$

is open and closed. The family $\{B_a\}_{a \in 2^*}$ forms a basis for the topology of 2^ω. Suppose $\{V_i\}_{i \in I}$ is an open cover of 2^ω. Then

$$T = \{\varepsilon\} \cup \{a \in 2^* \mid \forall i \in I \,.\, B_a \not\subseteq V_i\}$$

is a tree. Furthermore, T does not have an infinite path. To see this, consider any $\alpha \in 2^\omega$. There exists $i \in I$ such that $\alpha \in V_i$ and because B_a's form a basis, there exists $n \in \mathbb{N}$ such that $\alpha \in B_{\alpha_1 \ldots \alpha_n} \subseteq V_i$. But then $\alpha_1 \ldots \alpha_n \notin T$. By Weak König's Lemma T is a finite tree, therefore there exists $m \in \mathbb{N}$ such that $|a| < m$ for all $a \in T$. For every $b \in 2^*$ of length m there exists $i_b \in I$ such that $B_b \subseteq V_{i_b}$. The family $\{V_{i_b} \mid b \in 2^*, |b| = m\}$ is finite and covers 2^ω:

$$2^\omega = \bigcup_{|b|=m} B_b \subseteq \bigcup_{|b|=m} V_{i_b} \,.$$

It is not hard to see that the converse holds: Weak König's Lemma can be derived from compactness of 2^ω. We leave this as an exercise.

Side remark. We have not been precise about what set of axioms we used. Normally, this is not an important issue, as most mathematicians take for granted one of the standard axiomatizations of set theory, e.g., Zermelo-Fraenkel Set Theory. However, since both Weak König's Lemma and compactness of Cantor space are provable in Zermelo-Fraenkel set theory, we need to be precise about what it means for these two statements to be equivalent, lest we conclude that they are so simply because they are both true!

The proofs done in this section can be performed in a weaker set theory, such as Zermelo set theory[3] without the Axiom of Choice.[4] It turns out that the optimal setting for showing that Weak König's Lemma and compactness of 2^ω follow from each other is a much weaker system called RCA_0, which is a subsystem of Peano arithmetic. We do not go into further details here, the interested reader is referred to [Sim98]. The point we are making is that we cannot meaningfully talk about equivalence of two statements if our background theory proves them both.

[3] The main difference between Zermelo-Fraenkel and Zermelo set theory is that the latter does not have the Axiom of Replacement.

[4] We used choice for a finite family in this section (can you spot it?), but that one is provable from other axioms of set theory.

3 Kleene Tree

3.1 Computable functions

For background reading on computability I recommend [Odi89]. We assume some informal knowledge of how general-purpose computers work:

- A computer is a machine which executes basic instructions, such as "add 1 to the number stored in location 42".
- Instructions are executed one at a time in an orderly fashion.[5]
- Only finitely many instructions are executed in a finite amount of time.
- The behavior of a computer is determined by a program, which is a *finite* sequence of instructions.
- A computer may store input, output and intermediate data onto a tape, disk, memory or other storage device. It is assumed that the amount of storage is potentially infinite, i.e., we do not worry about computer running out of free storage.

An important observation is that there are only countably many programs, because there are finitely many basic instructions and a program is a finite sequence of basic instructions. Therefore, we may enumerate all programs P_0, P_1, P_2, ... systematically. In fact, there is a program which enumerates them all: such a program simply enumerates all finite sequences of basic instructions. It is beside the point that some programs do not compute anything useful, or that certain sequences of instructions do not make much sense.

We consider a program P that receives as input a natural number n. When P is executed, three things may happen:

- After a finite number of steps, P computes a result m, which is a natural number.
- After a finite number of steps, P blocks and does not produce an output.
- P runs forever without producing an output.

In the first case we say that P *terminates* on input n and write $P(n)\downarrow$. In the other two cases P is said to *diverge*, which we write as $P(n)\uparrow$. Thus every program computes a *partial* function $\mathbb{N} \rightharpoonup \mathbb{N}$.

Definition 3.1 *A* computable function *is a partial function* $\mathbb{N} \rightharpoonup \mathbb{N}$ *which is computed by some program.*

We denote by φ_n the computable function which is computed by the n-th program P_n. Every computable function appears in the enumeration φ_1, φ_2, φ_3, ..., possibly more than once because many different programs may compute the same function.[6] We use the notation $\varphi_n(m)\downarrow$ and $\varphi_n(m)\uparrow$ to indicate that $\varphi_n(m)$ is defined and undefined, respectively.

[5] Parallel computers may execute many instructions at once. This does not essentially increase their computing power, only efficiency.

[6] In fact, it can be shown that every computable function must appear infinitely often under mild and reasonable conditions on φ.

The enumeration φ is computable in the sense that there is a program U, called the *universal machine*, which accepts as input numbers n and m such that:

- if $\varphi_n(m) = k$ then $U(n, m)$ terminates with output k,
- if $\varphi_n(m)\uparrow$ then $U(n, m)$ diverges.

The main idea for U is to compute the instructions of P_n and then simulate what P_n would do on input m.

A *total* function is a partial function which is defined for all arguments. There are only countably many total computable functions, but in contrast with the partial computable functions they cannot be enumerated computably.

Theorem 3.2 *There is no computable enumeration of total computable functions.*

Proof. Cantor's diagonal argument may be employed here. Let $\theta_1, \theta_2, \ldots$ be any computable sequence of total computable functions. Then the function $f(n) = \theta_n(n) + 1$ is total and computable but does not appear in the sequence. Indeed, $f \neq \theta_k$ because $f(k) \neq \theta_k(k)$. ∎

One may wonder why we could not use a similar argument to define a *partial* computable function which is different from every other partial computable function in a given sequence. However, given a computable sequence $\theta_1, \theta_2, \theta_3, \ldots$ of partial computable functions the function $f(n) = \theta_n(n) + 1$ is *not* guaranteed to be different from $\theta(n)$ because we could have $\theta_n(n)\uparrow$ and so $f(n)$ and $\theta_n(n)$ would both be undefined. One might also try with

$$g(n) = \begin{cases} \theta_n(n) + 1 & \text{if } \theta_n(n)\downarrow, \\ 0 & \text{if } \theta_n(n)\uparrow, \end{cases}$$

which is a function different from every $\theta_1, \theta_2, \ldots$ Unfortunately, g is not computable in general, as we cannot determine in finite time whether a given computation terminates or not.

Yet, there is a good idea hidden in the above attempts. A slightly different diagonalization procedure gives us a computable partial function which is different from every total computable function.

Theorem 3.3 *There exists a computable partial function $d : \mathbb{N} \rightharpoonup \mathbb{N}$ such that whenever $f : \mathbb{N} \to \mathbb{N}$ is a total computable function, there exits $n \in \mathbb{N}$ for which $d(n)\downarrow$ and $f(n) \neq d(n)$.*

Proof. Define d by

$$d(n) = \begin{cases} 0 & \text{if } \varphi_n(n)\downarrow \text{ and } \varphi_n(n) \neq 0, \\ 1 & \text{if } \varphi_n(n)\downarrow \text{ and } \varphi_n(n) = 0, \\ \text{undefined} & \text{if } \varphi_n(n)\uparrow. \end{cases}$$

To compute $d(n)$, first compute $\varphi_n(n)$. If and when $\varphi_n(n)$ gives a result k, see whether $k = 0$. If it does, return 1, otherwise return 0. This shows that d is computable. If f is total there exists n such that $f = \varphi_n$. Because f is total $\varphi_n(n)$ is defined, therefore $d(n)$ is defined and $d(n) \neq \varphi_n(n) = f(n)$. ∎

3.2 Computable trees

We have defined computability for functions on natural numbers. Because infinite binary sequences are those functions on natural numbers which map into $\{0,1\}$, this gives as a notion of *computable binary sequence*. We denote the set of all computable binary sequences by $\#(2^\omega)$. Similarly, we let

$$\#B_a = B_a \cap \#(2^\omega)$$

be the computable part of the basic open set B_a. The family $\{\#B_a\}_{a\in 2^*}$ forms a countable basis for the topology of $\#(2^\omega)$.

We define a *computable set* $S \subseteq \mathbb{N}$ as one that has a computable characteristic function $\chi_S : \mathbb{N} \to 2$, which is defined by

$$\chi_S(n) = \begin{cases} 1 & \text{if } n \in S, \\ 0 & \text{if } n \notin S. \end{cases}$$

In the general case we may transfer computability from functions on natural numbers to other sets and structures by means of *Gödel encodings*. Suppose we want to speak about computability on a set S. We find a *representation* of elements of S by natural numbers, which is partial surjection $q : \mathbb{N} \rightharpoonup S$. When $q(n) = x$ we say that n *represents* or *encodes* element x. If we have two representations $q : \mathbb{N} \rightharpoonup S$ and $q' : \mathbb{N} \rightharpoonup T$, we say that a function $f : S \to T$ is *computable* if there exists a computable function $g : \mathbb{N} \rightharpoonup \mathbb{N}$ such that whenever $q(n) = x$ then $q'(g(n)) = f(x)$.[7]

The idea of representations is in fact quite familiar to every programmer. After all, in a digital computer *every* piece of data is encoded as a sequence of 0's and 1's. We have chosen to encode data in terms of numbers rather than binary sequences, but that is a minor detail.

Finite binary sequences are easily encoded as natural numbers. For example, we may encode the sequence $a_1 \dots a_n$ with the number $2^n 3^{a_1} 5^{a_2} \cdots p_{n+1}^{a_n}$, where p_i is the i-th prime number. Actually, it is better not to worry about encodings of finite binary sequences too much, since digital computers work the other way around: they encode numbers as finite binary sequences.

Since a tree is just a subset of 2^* we know what it means for it to be computable.

Definition 3.4 *A tree T is* computable *if the map*

$$a \mapsto \begin{cases} 1 & \text{if } a \in T, \\ 0 & \text{if } a \notin T \end{cases}$$

is computable.

Suppose T is an infinite computable tree. Then we may computably enumerate its leaves without repetition as follows. There is a computable enumeration of all elements of 2^*, for example

$$0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, \dots$$

For each element of this sequence we test whether it belongs to T, and remove those which do not.

[7] Which functions $S \to T$ are computable depends on the choice of representations q and q'. Thus it is important that we define reasonable representations.

3.3 Construction of Kleene tree

Let us now show that in the computable world König's Lemma fails quite badly.

Theorem 3.5 (Kleene [Kle52]) *There exists an infinite computable tree without a computable infinite path.*

Proof. Let D be a program which computes function d from Theorem 3.3. Note that the function d maps into $\{0,1\}$. Given an input n, we may execute $D(n)$ step by step. In principle, we do not know how long it will take to compute $D(n)$, or whether the computation will ever stop. But we may always abort the computation if it takes too long. So we define the k-th approximation to D to be the program

$$D^{(k)}(n) = \begin{cases} D(n) & \text{if } D(n) \text{ terminates in } \leq k \text{ steps,} \\ \texttt{abort} & \text{otherwise .} \end{cases}$$

The function $D^{(k)}$ is computable and total, even if D is partial.

The idea for Kleene tree is to put in it those binary sequences "that could be the sequence d". Since no infinite binary sequence can actually be equal to d, the tree will not contain any infinite paths. More precisely, let Kleene tree be

$$K = \{a \in 2^* \mid \forall 1 \leq k \leq |a| \,.\, (D^{(|a|)}(k) \neq \texttt{abort} \implies d(k) = a_k)\} \,.$$

We see that we put $a_1 \dots a_n$ in K when n computational steps are not sufficient to detect a difference between the prefix $d(0), d(1), \dots, d(n)$ and $a_1, a_2, \dots, a_n$. In other words, "a could be a prefix of d".

It is easy to see that K is a computable tree. Obviously $\varepsilon \in K$ and K is prefix-closed. The set K is computable: to compute whether $a \in K$ we compute for each $k = 1, \dots, |a|$ whether $D^{(|a|)}(k)$ outputs $\texttt{abort}$. If it does not, it outputs a 0 or a 1 which we then compare with a_k.

The tree K is infinite because it contains sequences of every length. Given any $n \geq 1$, the sequence $a_1 \dots a_n$ with

$$a_k = \begin{cases} d(k) & \text{if } D^{(n)}(k) \neq \texttt{abort}, \\ 0 & \text{otherwise,} \end{cases}$$

obviously is an element of K.

Finally, we show that every infinite computable sequence $\alpha \in \#(2^\omega)$ has a prefix which is not an element of K. Because α is computable there exists $j \in \mathbb{N}$ such that $\alpha = \varphi_j$. By Theorem 3.3 we have $d(j)\!\downarrow$ and $\alpha_j \neq d(j)$. Therefore $D(j)$ terminates within some number of steps, say m. But then $\alpha_1 \dots \alpha_m \notin K$ because $D^{(m)}(j) = d(j) \neq \alpha_j$. ∎

It is easy to see that $\#(2^\omega)$, viewed as a subspace of 2^ω is not compact because it is not a closed subspace: consider any $\alpha \in 2^\omega$ which is not computable and observe that it is the limit of a (non-computable) sequence $(\beta_n)_{n\in\mathbb{N}}$ with $\beta_n = \alpha_1 \dots \alpha_n 000 \cdots$. But this observation is not very illuminating from the point of view of computability theory. In computable analysis it is useful to know that $\#(2^\omega)$ fails to be compact in a computable way, i.e., that there exists a computable open cover such that for every finite subcover we can compute a point in $\#(2^\omega)$ which is not covered by the subcover. This we can do using Kleene tree.

Theorem 3.6 *There is a computable sequence $p : \mathbb{N} \to 2^*$ such that $\#(2^\omega)$ is covered by $\bigcup_{n\in\mathbb{N}} \#B_{p(n)}$ but every finite subcover of $\{\#B_{p(n)}\}_{n\in\mathbb{N}}$ fails to cover all of $\#(2^\omega)$. Moreover, there exists a computable map $f : \mathbb{N} \to \#(2^\omega)$ such that the finite subcover $\#B_{p(1)}, \ldots, \#B_{p(n)}$ does not contain $f(n)$.*

Proof. Let p be an enumeration of the leaves of Kleene tree K. Because every $\alpha \in \#(2^\omega)$ exits K, $\bigcup_{n\in\mathbb{N}} \#B_{p(n)}$ covers all of $\#(2^\omega)$.

Let $f(n)$ be the binary sequence which starts with $p(n+1)$ and continues with all 0's after that:

$$f(n)_k = \begin{cases} p(n+1)_k & \text{if } k \leq |p(n+1)|, \\ 0 & \text{otherwise} \end{cases}$$

Clearly, $f(n)$ is a computable binary sequence. Since p enumerates the leaves of K without repetition, it must be the case that $p(k) \not\sqsubseteq f(n)$ for all $k \leq n$, otherwise we would have $p(k) \sqsubseteq p(n+1)$ or $p(n+1) \sqsubseteq p(k)$, both of which are impossible because p enumerates the leaves of a tree without repetition. This shows that $f(n) \notin \#B_{p(1)} \cup \cdots \cup \#B_{p(n)}$. ∎

References

[Kle52] S.C. Kleene. Recursive functions and intuitionistic mathematics. In L.M. Graves, E. Hille, P.A. Smith, and O. Zariski, editors, *Proceedings of the International Congress of Mathemaiticans, August 1950. Cambridge, Mass.*, pages 679–685, 1952.

[Odi89] P. Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in logic and the foundations of mathematics*. North-Holland, 1989.

[Sim98] S.G. Simpson. *Subsystems of Second Order Arithmetic*. Springer-Verlag, 1998. ISBN 3-540-64882-8.