Different Problems, Common Threads: Computing the Difficulty of Mathematical Problems

Karen Lange

Mathematics is filled with existence theorems such as

Theorem 1. *Every vector space has a basis.*

Such statements do not address how one goes about *finding* the known-to-exist object. For example, Theorem 1 naturally leads to the "Basis Problem."

Problem 2 (Basis Problem)**.** Given a vector space, can we compute a basis for it?

It turns out that the answer to this question is no—from a computational viewpoint, bases may not exist for vector spaces. For this reason, we say that the Basis Problem is not a "computable" problem. We can, however, ask just how far from computable the Basis Problem is and what other problems have the same computational power. A natural way to compare the algorithmic difficulty of two problems is to determine whether having the ability to solve one allows a person to solve the other. In this case, the latter problem must be no "harder" than the former. Under this problem-reduction approach, two problems have the same computational power if they each can be used to solve the other.

In my talk, we will explore the key ideas behind taking a "computable" perspective on mathematics and how this compares with an "existence" one, as in the example above. We will also illustrate the problem-reduction approach using problems from across mathematics. Moreover, this approach has strong connections to proof theory, specifically, calibrating exactly which axioms of mathematics are needed to prove the original existence theorems.

My talk draws on ideas from computability theory (also known as recursion theory), a branch of logic. To learn more about what logic offers other areas of mathematics, check out the AMS-ASL Special Session, Logic Facing Outwards, Wednesday, January 15, 2:15PM–6PM, and Thursday, January 16, 8AM–12PM. These talks, like mine, will not assume a background in logic. The remainder of this article gives a taste of some of the ideas mentioned earlier.

To address the Basis Problem, one first needs a precise understanding of what it means to "compute." Alan Turing gave the accepted definition [4] in terms of so-called "Turing Machines," and his model of computation laid the theoretical framework for the invention of computers as we know them today. We will rely here on the useful (and accurate) anachronism of thinking of Turing's model as being a computer and Turing Machines as being computer programs that use natural numbers for inputs and outputs. In short, a Turing Machine is a finite set of instructions that can be applied to a particular input n in a step-by-step and completely determined fashion. Given some input n, the resulting procedure either terminates or "halts" after some finite number of steps and gives an output, or the computation continues to proceed forever. We can now define when a function with domain and codomain $\mathbb{N}$ is computable.

Definition 3. A function $f : \mathbb{N} \to \mathbb{N}$ is *computable* if there is a Turing Machine that, when run on input n, halts and outputs $f(n)$.

As long as a function's domain and codomain can be encoded by natural numbers, we can also discuss the computability of the function relative to that encoding. For example, it is not hard to come up with an easy-to-compute bijection between $\mathbb{N}^2$ and $\mathbb{N}$ that encodes pairs (i, n) as single natural numbers $\langle i, n\rangle$. Furthermore, a computer program written in a particular language can be viewed as a finite sequence of symbols from a fixed finite alphabet. By ordering all such finite sequences (say by length and then lexicographically), we obtain a list of programs $(P_i)_{i\in\mathbb{N}}$. Many of the programs P_i are nonsense (e.g., do not obey syntax rules), but certainly every possible program written in that language appears. Since there are only countably many programs, and thus Turing Machines, and uncountably many functions $f : \mathbb{N} \to \mathbb{N}$, we know that there *exist* noncomputable functions. Even better, though, the encoding $(P_i)_{i\in\mathbb{N}}$ allows us to write down a particular example of a function that is not computable.

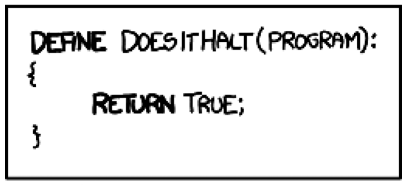
Consider the *Halting Problem*:

Problem 4 (Halting Problem)**.** If program P_i is run on input n, does the computation halt?

In other words, the problem asks whether a particular program's computation terminates when run on a particular input (see Figure 1). The Halting Problem is likely all too familiar. Computer users encounter it when they see the spinning "wheel" or "beachball of death" on their screen

Karen Lange is an associate professor of mathematics at Wellesley College. Her email address is karen.lange@wellesley.edu.

DOI: https://doi.org/10.1090/noti2001



THE BIG PICTURE SOLUTION TO THE HALTING PROBLEM

Figure 1. "Halting Problem" by Randall Munroe.

Figure 2. "Beachball of death" by Mark H. Evans.

and have to decide whether to wait or restart the machine (see Figure 2).

We can encode a "solution" to the Halting Problem by the function $h : \mathbb{N} \to \mathbb{N}$ where

$$h(\langle i, n\rangle) = \begin{cases} 1 & \text{if program } P_i \text{ run on input } n \text{ halts} \\ & \text{after finitely many steps,} \\ 0 & \text{otherwise.} \end{cases}$$

One can show that the function h, and thus the Halting Problem, is not computable.

Notice that the Halting Problem is an existence statement in disguise: the problem is asking whether there is some point in time when the computation is finished. In fact, the Halting Problem encodes all purely existential information in a sense that can be made precise and so plays a central role in computability theory.

Let us return to the problem of computing a basis for a given vector space. Although we have formalized what we mean by computation, the Basis Problem remains imprecise without making explicit the terms "given a vector space" and "compute a basis." If we are to "compute a basis," it seems reasonable that we must be able to compute information about the given vector space. As alluded to above, an object can only be computable if it can be encoded in terms of natural numbers. Hence, we will assume we are working with countable vector spaces, say over the countable field $\mathbb{Q}$ for concreteness. (This assumption is standard, but there are ways to discuss the computability of uncountable structures; see, e.g., [2].) Once an indexing $(\vec{v}_i)_{i\in\mathbb{N}}$ of a vector space V is fixed, the vector addition and scalar multiplication by any given $q \in \mathbb{Q}$ on V induce functions on the indices i. In other words, the induced functions, which we will call i_+ and $i_{q\cdot}$, describe the vector space operations in terms of the indexing (see Figure 3 for a careful definition of i_+ and $i_{q\cdot}$).

Definition 5. Let V be a countable vector space over $\mathbb{Q}$.

1. A *presentation of* V is an indexing $(\vec{v}_i)_{i\in\mathbb{N}}$ of the elements of V together with the functions i_+ and $i_{q\cdot}$ for each $q \in \mathbb{Q}$.
2. A presentation $(\vec{v}_i)_{i\in\mathbb{N}}$ of V is *computable* if the functions describing vector addition and scalar multiplication in terms of this indexing are computable; i.e., i_+ and the function $(q, n) \to i_q(n)$ are computable.

Vector Addition on Indices	Scalar Multiplication by $q \in \mathbb{Q}$
$i_+ : \mathbb{N}^2 \to \mathbb{N}$	$i_{q\cdot} : \mathbb{N} \to \mathbb{N}$
$(i, j) \to k$ if	$n \to m$ if
$\vec{v}_i + \vec{v}_j = \vec{v}_k$	$q\vec{v}_n = \vec{v}_m$

Figure 3. Definitions of the functions i_+ and $i_{q\cdot}$.

Thus, when we say "given a vector space" in the Basis Problem, we mean that we are handed a computable presentation $(\vec{v}_i)_{i\in\mathbb{N}}$ of a vector space. The phrase "compute a basis" is then synonymous with giving an algorithm for determining which $\vec{v}_i$ in V are in the basis. In other words, a basis relative to a presentation $(\vec{v}_i)_{i\in\mathbb{N}}$ is computable if the characteristic function (in terms of the indices i) of the basis is computable. We can now formalize the Basis Problem as

Problem 6. Does every computable presentation of a (countable) vector space have a computable basis?

We mentioned earlier that the answer is no. One can construct such a vector space by building a vector space in which certain linear dependences do not become apparent until late in the construction. Although the Basis Problem is not computable, it is no harder than the Halting Problem. Given a computable vector space V, the ability to solve the Halting Problem allows one to compute a basis for V. Even more exciting, the Basis Problem can be used to solve the Halting Problem, in that there is a computable presentation of a vector space V such that any basis of V can be used to compute the function h. Therefore, the two problems are computationally the same [1]; see also [3, §III.4].

To fully understand the power of these problems, we need to compare them to others. A version of the Heine–

Borel theorem for the real closed unit interval (i.e., finding a finite subcovering of an open covering made of intervals) is a strictly weaker problem than the Halting/Basis Problems, whereas writing a countable abelian group as the direct sum of a divisible group and a reduced group is a strictly harder one [3, §§IV.1, VI.4]. The overall structure of problem difficulty is extremely rich, and understanding which problems have the same computational power illuminates what makes these problems "tick." Please join me as we take a computable perspective on mathematics at the 2020 Joint Meetings. I also hope to see you at the AMS-ASL Special Session, Logic Facing Outwards, Wednesday, January 15, and Thursday, January 16.

References

[1] Friedman HM, Simpson SG, Smith RL. Countable algebra and set existence axioms, *Ann. Pure Appl. Logic*, no. 2 (25):141–181, 1983. https://0-doi.org.luna.wellesley.edu/10.1016/0168-0072(83)90012-X. MR725732

[2] Miller RG. Locally computable structures. *Computation and Logic in the Real World*, Springer, Berlin, 2007, 575–584. https://0-doi.org.luna.wellesley.edu/10.1007/978-3-540-73001-9_60 MR2646269

[3] Simpson SG. *Subsystems of Second Order Arithmetic*, Second edition, Perspectives in Logic, Cambridge University Press, Cambridge; Association for Symbolic Logic, Poughkeepsie, NY, 2009. https://0-doi.org.luna.wellesley.edu/10.1017/CBO9780511581007 MR2517689

[4] Turing AM. On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Math. Soc. (2)*, no. 3 (42):230–265, 1936. https://0-doi.org.luna.wellesley.edu/10.1112/plms/s2-42.1.230. MR1577030

Karen Lange

Credits

Figure 1 is by Randall Munroe, licensed under CC BY-NC 2.5 (see https://creativecommons.org/licenses/by-nc/2.5), and available at https://xkcd.com/1266.

Figure 2 is by Mark H. Evans, licensed under CC BY 2.0 (see https://creativecommons.org/licenses/by/2.0), and available at https://www.flickr.com/photos/mevs/5306313081/in/photostream.

Photo of the author is courtesy of Richard Howard.