Philosophy of Science (2023), **90**, 1079–1088
doi:10.1017/psa.2023.26

CONTRIBUTED PAPER

Mathematical Explanation in Computer Science

André Curtis-Trudel

Lingnan University, Hong Kong
Email: andre.curtistrudel@ln.edu.hk

(Received 23 January 2023; accepted 24 January 2023; first published online 16 February 2023)

Abstract

This note scouts a broad but underexplored class of explanations found in contemporary computer science. These explanations, which I call limitative explanations, explain why certain problems cannot be solved computationally. Limitative explanations are philosophically rich, but have not received the attention they deserve. The primary goals of this note are to isolate limitative explanations and provide a preliminary account of what makes them explanatory. On the account I favor, limitative explanations are a kind of non-causal mathematical explanation which depend on highly idealized models of computation.

1. Introduction

This note scouts a broad class of explanations of central importance to contemporary computer science. These explanations, which I call limitative explanations, explain why certain problems cannot be solved computationally, either in principle or under certain constraints on computational resources such as time or space. Limitative explanations are philosophically rich, but have not received the attention they deserve. The primary goals of this note are to isolate limitative explanations and provide a preliminary account of their explanatory power. On the account I favor, limitative explanations are a kind of non-causal mathematical explanation which depend on highly idealized models of computation.

Here is the plan. Section 2 surveys some familiar results from theoretical computer science. Section 3 argues that these results are prima facie explanatory of certain features of physical computing systems. Sections 4 through 6 sketch a positive account of their explanatory power. Section 7 concludes.

2. Limitative results in computer science

At the broadest level, computer science addresses two kinds of problem: those which can be solved computationally, and those which cannot. To address the first kind of problem, we must describe a computational procedure that solves it. To address the second kind, by contrast, we must show that no such procedure exists. This typically involves proving a mathematical theorem to the effect that the problem in question cannot be solved computationally. For want of a name, I shall call results like this "limitative results."

Perhaps the most well-known limitative results are impossibility theorems, which state that no general computational procedure exists for a certain class of problems, even in principle. One famous result of this sort is the unsolvability of the halting problem. Informally, this is the problem of determining whether a given Turing machine halts on an arbitrary input. Notoriously, of course, no Turing machine solves this problem (Turing 1936). This result thus captures a limit on the computational powers of Turing machines: even given unlimited computational time and space, they simply cannot solve the halting problem.

Similar results play a central role in many branches of contemporary computer science, including algorithmic analysis (Sedgewick and Flajolet 2013), distributed systems theory (Attiya and Ellen 2014), artificial intelligence (Minsky and Papert 1988), and computer security (Cohen 1987), among others. To illustrate with a perhaps less familiar example, consider the following problem from the field of compiler design. A compiler is a program for translating programs written in one programming language into another. One basic requirement is that a compiler preserve the input/output profile of the program being translated. Often, however, to improve performance, a compiler will attempt to optimize the translated program, for instance by detecting and eliminating needlessly duplicated instructions. A *fully optimizing compiler* is one which, given some program as input, produces as output the smallest possible program with the same input/output profile as the original. The fully optimizing compiler problem is the problem of writing such a compiler.

Perhaps surprisingly, it can be shown that no fully optimizing compiler exists (Appel and Palsberg 2002, ch. 17). The reason is that the problem of fully optimizing a program is equivalent to the halting problem. For consider a program which never halts. The smallest possible program with the same input/output profile is:

```
L: goto L
```

Given any input, this program immediately goes into an infinite loop. Thus, to detect whether an arbitrary program is input/output equivalent to this one, a compiler would have to be able to detect whether a program ever halts. However, if it could do this, it could in effect solve the halting problem. Since this is impossible, it is impossible to write a fully optimizing compiler.

Impossibility theorems are the clearest and most straightforward kind of limitative result, and for this reason they will be the primary focus of my discussion. For completeness' sake, however, let me briefly mention two others. First, while impossibility theorems identify limits on what can be computed in principle, other limitative results identify what can be computed *tractably*, using only a bounded amount of computational time or space. For example, some problems are known to be solvable only in time exponential in the size of their inputs, and identifying lower bounds on the resources required to solve a problem is an important part of computational complexity theory (Papadimitriou 1994, ch. 20).

Another kind of limitative result identifies tradeoffs between different solutions to a problem. For instance, one technique for dealing with a problem known or strongly

https://doi.org/10.1017/psa.2023.26 Published online by Cambridge University Press

suspected to be intractable is to attempt to find good (albeit suboptimal) solutions through optimization or approximation techniques. Investigation into such techniques has identified a family of results known colloquially as "no free lunch" theorems (Wolpert and Macready 1997). These identify tradeoffs between different search or optimization strategies, in the sense that improved performance for some range of problems is offset by decreased performance for others. These results thus identify limits on approximate solutions to certain problems.

3. From limitative results to limitative explanations

Limitative results are, in the first instance, mathematical theorems. They are characterized in terms of the members of a class of formal, mathematically characterized computational models such as Turing machines. Consequently, limitative results primarily concern the computational powers of these mathematically characterized computational models; the halting problem tells us something about what *Turing machines* in particular cannot do.

Nonetheless, these results are widely taken to bear on the computational powers of physical computing systems as well. This is because Turing machines are taken to be models of physical computing systems, such as general-purpose stored-program computers (Savage 2008). Consequently, when a system can be accurately modeled as a Turing machine, it is plausible to think that limits on what can be computed by Turing machines apply to that system as well.

Among philosophers, the idea that certain physical systems can be "accurately described" as Turing machines is normally cashed out in terms of computational implementation. Roughly speaking, a physical system implements a computational model if that model captures the basic computational architecture of that system. This involves, among other things, capturing its basic computational operations and memory structure. What model(s) a system implements determines which computational problems it can solve: crudely, it can solve (at most) those problems solvable by the model in question.

It is relatively uncontroversial that limitative results describe limits on the computational powers of physical computing systems. However, I think that they do more than this. Sometimes, they *explain* those limits as well. This attitude is common in computer science. For instance, a popular introductory monograph on computational complexity begins with the following parable:

> One day your boss calls you into his office and confides that the company is about to enter the highly competitive "bandersnatch" market. For this reason, a good method is needed for determining whether or not any given set of specifications for a new bandersnatch component can be met and, if so, for constructing a design that meets them. Since you are the company's chief algorithm designer, your charge is to find an efficient algorithm for doing this.
>
> . . . Some weeks later, your office filled with mountains of crumpled-up scratch paper, your enthusiasm has lessened considerably. So far you have not been able to come up with any algorithm substantially better than searching through all possible designs. This would not particularly endear you to your boss, since it

https://doi.org/10.1017/psa.2023.26 Published online by Cambridge University Press

> would involve years of computation time for just one set of specifications, and the bandersnatch department is already 13 components behind schedule.
>
> . . . To avoid serious damage to your position within the company, it would be much better if you could prove that the bandersnatch problem is *inherently* intractable, that no algorithm could possibly solve it quickly. You could then stride confidently into the boss's office and proclaim: "I can't find an efficient algorithm because no such algorithm is possible!" (Garey and Johnson 1979, 1–2)

This "because" is no accident. The fact that no efficient bandersnatch algorithm exists explains why one's attempts to find one fail. Similarly, the fact that the halting problem is unsolvable would seem to explain why my laptop, or indeed *any* system that can be modeled as a Turing machine, fails to solve it. When limitative results are used in this way, I call them "limitative explanations."

Although more could be said in defense of the claim that limitative results are explanatory, for now I will take this claim for granted to see where it takes us. If limitative results are explanatory, where does their explanatory power come from? Two observations guide my analysis. First, mathematical theorems play a central role in limitative explanations. An adequate account of their explanatory power should address this role. Second, limitative results apply extraordinarily widely. Indeed, Turing's result is striking partly because it applies to systems as disparate as effective human workers, contemporary digital computers, and even to certain unconventional computing systems such as cellular automata and quantum computers. Yet despite their architectural differences, none of these systems can solve the halting problem. Turing's result seems to explain this, and so an adequate account of limitative explanations should thus say something about how this is so.

4. Limitative explanations as mathematical explanations

Because they rely on mathematical theorems, a natural starting point is the idea that they are a kind of mathematical scientific explanation. Although there are a few different accounts of mathematical explanation in the literature, I will develop this idea with reference to a well-known account due to Lange (2017, 2018). Explanations by constraint, as Lange sometimes calls them, work by "describing how the explanandum involves stronger-than-physical necessity by virtue of certain facts ('constraints') that possess some variety of necessity stronger than ordinary causal laws" (Lange 2018, 17). Although the explananda of such explanations can come in varying modal strengths, they are in general modally stronger than the explananda of typical causal explanations. Accordingly, their explanans must involve facts of equal or greater modal strength. Mathematical explanations in particular are a kind explanation by constraint in which at least one component of the explanans is mathematically necessary.

To illustrate with a well-known case, consider the bridges of Königsberg. Crudely, the reason why no one has successfully crossed all of the bridges without crossing at least one of them more than once is that (a) the bridges realize a certain graph-theoretic structure, a non-Eulerian graph, and (b) as a matter of mathematical necessity any complete circuit of a non-Eulerian graph has at least one

https://doi.org/10.1017/psa.2023.26 Published online by Cambridge University Press

double-crossing (Pincock 2007). Here the explanans, that no one has successfully crossed the bridges in a certain way, is modally stronger than ordinary causal laws such as the force laws. Even if we could change the force laws, there would still be no successful crossing. Thus, to explain this we must appeal to something modally stronger—in this case, a mathematical necessity.

Similarly, it would seem that the reason why my laptop fails to solve the halting problem is that (a) it has a certain computational structure, of the sort roughly captured by a Turing machine, and (b) it is mathematically necessary that no object with that structure can solve the halting problem. As with the bridges case, this explanandum is more necessary than ordinary causal laws: vary the force laws, and my laptop still wouldn't solve the halting problem. Thus, to explain this fact we need something modally stronger. Turing's theorem that the halting problem is unsolvable fits the bill.

Elsewhere, Lange suggests that explanations by constraint concern the "framework" in which more ordinary causal explanations operate. Such explanations work "not by describing the world's actual causal structure, but rather by showing how the explanandum arises from the framework that any *possible* physical system ... must inhabit" (Lange 2017, 30). This is unlike ordinary causal explanation, which takes for granted a certain framework—e.g., as captured by the force laws—in which causes operate. Explanations by constraint, by contrast, arise from the framework underwriting causal explanation. They concern what must (or cannot) be the case, given that framework.

Although limitative explanations clearly do not rely on claims about the framework that *any* possible physical system must inhabit, it is not implausible to think that they concern general facts about the frameworks underwriting *causal* computational explanations in particular. To see this, consider explanation via program execution, an uncontroversially causal kind of computational explanation (Piccinini 2015, ch. 5). These take for granted a stock of primitive computational operations (typically basic logical or arithmetical operations), and explain by citing a step-by-step causal process composed of these basic operations. For instance, the reason why my laptop displays these words on the screen is that it executes a particular word-processing program transforming key-presses into pixel patterns.

A different kind of computational explanation comes into view, however, when we consider a fixed set of computational operations and resources—for example, by focusing on a fixed programming language—and then ask what problems can be solved given that set. This leads us towards limitative explanation. By showing that some problem cannot be solved using those computational operations and resources, we thereby show that no computational model—for example, no specific program written in a fixed language—that uses only those operations and resources can solve that problem either.

These points suggest that limitative explanations can be profitably understood as a kind of mathematical explanation. However, this isn't the whole story. Merely noting that limitative explanations cite a mathematical constraint does not obviously address their wide applicability. Even though it is mathematically necessary, why not think that Turing's theorem applies only to a specific, restricted class of physical computing systems, rather than more broadly? I take up this question next.

https://doi.org/10.1017/psa.2023.26 Published online by Cambridge University Press

5. Essential idealization

My account of the wide applicability of limitative results proceeds in two steps. To keep things simple, I start by looking more closely at the implementation relation connecting Turing machines to a single class of physical computing systems, namely human agents working effectively. I argue that limitative results apply to effective human agents only under significant idealization. Then, in the next section, I will consider how to extend this basic story to other kinds of computing systems.

Recall that a physical system implements a computational model if that model captures the computational architecture of that system, such as its basic computational operations and memory organization. Because different computational models take different operations as primitive and have different memory structures, implementation conditions differ between models. For instance, consider a standard deterministic Turing machine (DTM). DTMs are equipped with a read/write head and a one-dimensional tape. They manipulate symbols one at a time on the tape according to a finite, predetermined set of instructions. Thus, a physical system implements a DTM if, roughly speaking, it manipulates symbols one at a time according to a finite set of determinate instructions on a one-dimensional tape.

So construed, however, few physical systems literally implement DTMs. Few contemporary digital computers, for instance, manipulate symbols one at a time, nor do they manipulate these symbols by scanning back and forth across a one-dimensional array. Indeed, among physical systems that might reasonably be construed as DTMs, human agents working effectively are perhaps the most plausible example. This is of course unsurprising given that Turing's characterization aimed, in the first instance, to capture the basic elements of effective human calculation (Sieg 2009). However, even here the differences are substantial. Whereas DTMs are assumed to never break down, to follow instructions perfectly, and to have an infinitely long tape (or at least potentially infinite), humans working effectively may fail to follow instructions correctly, only have a finite amount of memory to work on (there is only so much scratch paper in a finite universe), and, alas, will in the fullness of time break down.

These observations suggest that humans working effectively implement DTMs only under significant idealization. This is in some respects a familiar point. Quite often we must idealize to bring mathematics to bear on physical systems (Bueno and French 2018). However, as I will argue next, what is striking about the computational case is that absent these idealizations limitative results do not obviously apply to physical computing systems. These idealizations are thus essential to limitative explanations.

First, notice that actual human calculators have access to only a finite amount of memory. In fact, they are more literally described as Turing machines with only a finite amount of tape, sometimes known as bounded tape Turing machines (BTTMs). Human agents working effectively implement full-strength DTMs only under the idealizing assumption that they have unbounded memory. Now consider an in-principle unsolvable problem such as the halting problem. In practical terms, the unsolvability of the halting problem ensures that no human calculator can determine whether a given algorithm will halt on a given input. It turns out, however,

https://doi.org/10.1017/psa.2023.26 Published online by Cambridge University Press

that this problem is unsolvable only under the idealization that human calculators have unbounded memory resources.

To see this, consider the *bounded* halting problem. This is the problem of determining whether a machine with only a finite, pre-determined amount of memory halts on a given input. This problem is decidable, because a deterministic system with bounded memory has only finitely many possible configurations (i.e., combinations of internal states and memory contents). Thus, we can let the system run until either it produces the desired output, or it goes into a previously seen configuration. In the latter case, because the system is deterministic we know that the system has entered an infinite loop, and so can determine that it will never halt (Sipser 2013, p. 222, Theorem 5.9).

There is thus a sense in which the halting problem doesn't even *concern* ordinary human calculators. If any version of the halting problem applies to them, it's the bounded halting problem. But that problem is decidable. If we knew how much memory they had at their disposal, we could determine whether a human calculator following an effective procedure would halt on a given input. But if this is right, then it's unclear why we should take limitative results such as the halting problem to apply to human calculators in the first place. And if these results do not even apply to such systems, it's hard to see how they can explain anything about them either.

So why, despite this, do computer scientists continue to use DTMs rather than, say, BTTMs to model physical computing systems? Crudely put, the reason is that DTMs reveal deeper facts about their computational powers. Consider again effective human workers. The fact that they only have finite memory is not so much a reflection of their basic computational capacities as a contingent fact about the kind of world they happen to find themselves in. If effective human workers *did* have unbounded memory resources, Turing machines would much more closely approximate them and the traditional halting problem would more straightforwardly apply. The problem is "merely" that the world doesn't cooperate, as it were.

Another way to put the point is that idealizations that provision more memory do not require that we change the basic computational operations carried out in effective human calculation (namely, finite operations on bounded data structures). For this reason, they allow us to clarify what kinds of problems can and cannot be solved using such operations. Contrast this with idealizations concerning basic computational operations, for instance by allowing infinitary instructions or architectures in which successive operations are executed twice as quickly. These idealizations depart much more drastically from effective human calculation. Although a human working effectively may in some sense, and under significant idealization, implement an infinitary computational model, it is much less clear that characterizing human calculation this way reveals much of interest regarding the powers of actual human calculation.

6. Simulation equivalence

The next step is to extend this story to explain how limitative results apply as widely as they do. Here the problems are of a rather different character. Earlier I noted that the DTM architecture differs significantly from the architecture found in many actual computing systems. Idealizations notwithstanding, why should we think that

https://doi.org/10.1017/psa.2023.26 Published online by Cambridge University Press

limitative results framed in terms of DTMs apply to systems with widely different computational architectures?

To bring out the problem, consider a contemporary microprocessor. Like a DTM, a microprocessor has memory and processing unit, and manipulates finitely many digits at a time. But the similarities end there (plus or minus a few details). Unlike a Turing machine, the physical system's workspace is broken up into different components (registers, RAM, storage, etc.), it operates directly on 32- or 64-bit words, and its datapath is typically highly parallelized, to name just a few differences (Harris and Harris 2013; Hennessy and Patterson 2003).

Indeed, contemporary microprocessors are much more accurately described as register machines than Turing machines. Register machines are an idealized representation of the von Neumann architecture employed in many contemporary digital computers. Whereas a DTM has a single contiguous block of one-dimensional memory, register machines are equipped with a bank of discrete memory locations ("registers"). And whereas DTMs take certain string-theoretic operations as primitive (e.g. reading, erasing, and writing individual symbols), register machines typically take as primitive basic logical and arithmetical operations on register contents. In light of these architectural similarities, it is much more natural to think that microprocessors implement register machines (under appropriate idealizations) than Turing machines.

In spite of this, many limitative results framed in terms of DTMs are nonetheless taken to apply to contemporary microprocessors. How do we reconcile this with the fact that microprocessors are more accurately described as register machines? The answer is that DTMs are, in a sense to be explained shortly, computationally just as powerful as register machines. Thus, limits on the computational powers of DTMs carry over to register machines and thereby to their implementations.

The standard technique for showing that two computational models M_1 and M_2 are computationally equally powerful is to show that any given procedure framed in terms of one can be simulated by the other. This involves demonstrating how to systematically transform an M_1-computation into an M_2-computation, and vice versa. In practice, this is facilitated by first proving a universality theorem, which identifies a single machine which can solve any problem solvable by a given model. For example, Turing discovered a universal Turing machine capable of solving any problem solvable by a DTM. With a pair of universality theorems in hand, one for each model in question, we need only show how to transform computations carried out by one universal machine in terms of the other. When two computational models are equivalent in this sense, I will say that they are simulation equivalent.

Simulation equivalent models can solve exactly the same computational problems. For suppose we know how to solve a problem with an M_1 machine. Then, given the simulation equivalence of M_1 and M_2, we can systematically transform the M_1-solution into an M_2-solution. Similarly, if no M_1 machine solves some problem, then no M_2-solution exists either. For suppose not. By their simulation equivalence, we could transform the M_2-solution into an M_1-solution, a contradiction.

Computational models simulation equivalent to Turing machines are said to be *Turing complete*. Many computational models are known to be Turing complete, including register machines, and most contemporary programming languages. Because all of these models are computationally equipowerful, a limitative result framed in terms of one of them applies to the others as well.

https://doi.org/10.1017/psa.2023.26 Published online by Cambridge University Press

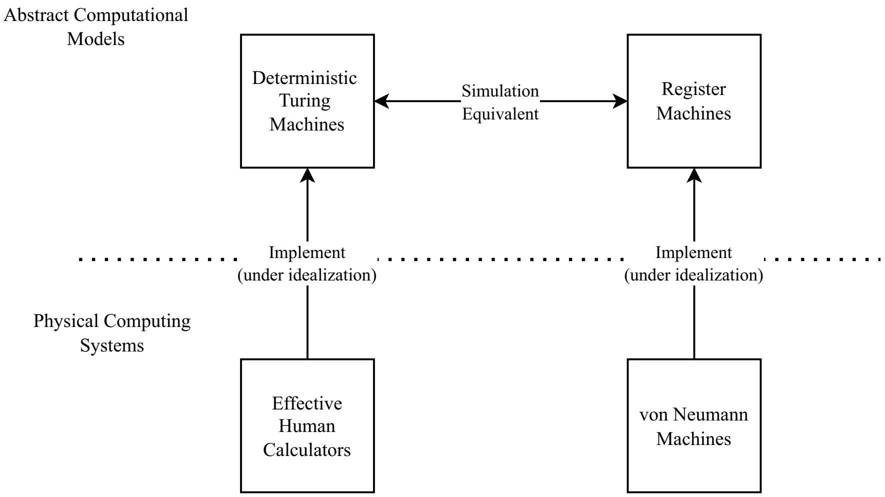
Abstract Computational Models
Deterministic Turing Machines
Simulation Equivalent
Register Machines
Implement (under idealization)
Implement (under idealization)
Physical Computing Systems
Effective Human Calculators
von Neumann Machines

Figure 1. How limitative results apply.

I am now in a position to explain how limitative results apply so widely. Different kinds of physical computing systems directly implement different kinds of computational models. Which computational model a given system implements depends on its architectural features: its primitive operations, memory organization, and so forth. For instance, humans working effectively implement DTMs, while digital computers implement register machines. Limitative results framed in terms of one kind of computational model apply to implementations of a different kind of computational model just in the case that the two models are simulation equivalent. Metaphorically, we can think of the explanatory power of a limitative result flowing outward from a node in a network: it applies to different computational models through relations of simulation equivalence, and projects down to physical computing systems via relations of computational implementation. (See Figure 1 for a partial sketch of the situation.)

7. Next steps

The account sketched in this paper raises certain interesting issues regarding computational explanation. Let me close by mentioning two. First, even if DTMs, register machines, and so forth are all in some sense explanatory, one wonders whether limitative results framed in terms of different models display different explanatory virtues. For instance, one might think that highly abstract models such as the λ-calculus, partial recursive functions, or recursively enumerable sets provide "deeper" explanations than those framed in terms of register machines, whereas explanations framed in terms of register machines "purer" in the sense that they more directly capture computational limits on specific kinds of physical systems. If so, there are important explanatory tradeoffs to be considered when employing one computational model over another.

https://doi.org/10.1017/psa.2023.26 Published online by Cambridge University Press

Second, dominant views of computational explanation treat it by and large as a kind of causal explanation. Roughly, on these views, computational explanations are causal explanations whose relata are *computational* states, events, processes, etc. (Piccinini 2015). However, if limitative explanations are a kind of explanation by constraint, and if explanations by constraint are non-causal, so too are limitative explanations. If, moreover, limitative explanations are genuine computational explanations, they would thus appear to be a kind of non-causal computational explanation hitherto unaccounted for by dominant theories of computational explanation.

Acknowledgments. Thanks to Soyeong An, Zoe Ashton, Chris Pincock, Lisa Shabel, Richard Samuels, Stewart Shapiro, John Symons, and audiences at Western University and the Method, Theory, and Reality workshop for questions and discussion. Work on this paper was partially supported by a Senior Research Fellowship award from the Research Grants Council of the Hong Kong SAR, China ("Philosophy of Contemporary and Future Science," Project no. SRFS2122-3H01).

References

Appel, Andrew W. and Jens Palsberg. 2002. *Modern Compiler Implementation in Java*. Cambridge: Cambridge University Press.

Attiya, Hagit and Faith Ellen. 2014. *Impossibility Results for Distributed Computing*. San Rafael, CA: Morgan & Claypool Publishers.

Bueno, Otávio and Steven French. 2018. *Applying Mathematics: Immersion, Inference, Interpretation*. Oxford: Oxford University Press.

Cohen, Fred. 1987. "Computer Viruses." *Computers & Security* 6 (1):22–35.

Garey, Michael R. and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman.

Harris, David M. and Sarah L. Harris. 2013. *Digital Design and Computer Architecture*, 2nd ed. Burlington, MA: Morgan Kaufmann.

Hennessy, John L. and David A. Patterson. 2003. *Computer Architecture: A Quantitative Approach*. Burlington, MA: Morgan Kaufmann.

Lange, Marc. 2017. *Because Without Cause: Non-Causal Explanations in Science and Mathematics*. Oxford: Oxford University Press.

Lange, Marc. 2018. "Because Without Cause: Scientific Explanations by Constraint." In *Explanation Beyond Causation: Philosophical Perspectives on Non-Causal Explanations*, edited by Alexander Reutlinger and Juha Saatsi, 15–38. Oxford: Oxford University Press.

Minsky, Marvin and Seymour Papert. 1988. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA: MIT Press.

Papadimitriou, Christos H. 1994. *Computational Complexity*. Boston, MA: Addison-Wesley.

Piccinini, Gualtiero. 2015. *Physical Computation: A Mechanistic Account*. Oxford: Oxford University Press.

Pincock, Christopher. 2007. "A Role for Mathematics in the Physical Sciences." *Nous* 41 (2):253–275.

Savage, John E. 2008. "Models of Computation: Exploring the Power of Computing." Available at https://cs.brown.edu/people/jsavage/book/.

Sedgewick, Robert and Philippe Flajolet. 2013. *An Introduction to the Analysis of Algorithms*. Boston, MA: Addison-Wesley. Second edition.

Sieg, Wilfried. 2009. "On Computability." In *Philosophy of Mathematics*, edited by Andrew Irvine, 535–630. Amsterdam: North-Holland.

Sipser, Michael. 2013. *Introduction to the Theory of Computation*. Boston, MA: Cengage.

Turing, Alan. 1936. "On Computable Numbers, with an Application to the Entscheidungsproblem." *Proceedings of the London Mathematical Society* 42 (1):230–265.

Wolpert, David H. and William G. Macready. 1997. "No Free Lunch Theorems for Optimization." *IEEE Transactions on Evolutionary Computation* 1 (1):67–82.

Cite this article: Curtis-Trudel, André. 2023. "Mathematical Explanation in Computer Science." *Philosophy of Science* 90 (5):1079–1088. https://doi.org/10.1017/psa.2023.26