Q & A

How to Write Software With Mathematical Perfection

By SHEON HAN

May 17, 2022

Leslie Lamport revolutionized how computers talk to each other. Now he's working on how engineers talk to their machines.

1 |

Modern computers can effectively coordinate with each other because of the work of the computer scientist Leslie Lamport. He's since turned his attention to making programming itself more efficient.

—

Talia Herman for Quanta Magazine

Leslie Lamport may not be a household name, but he's behind a few of them for computer scientists: the typesetting program LaTeX and the work that made cloud infrastructure at Google and Amazon possible. He's also brought more attention to a handful of problems, giving them distinctive names like the bakery algorithm and the Byzantine Generals Problem. This is no accident. The 81-year-old computer scientist is unusually thoughtful about how people use and think about software.

In 2013, he won the A.M. Turing Award, considered the Nobel Prize of computing, for his work on distributed systems, where multiple components on different networks coordinate to achieve a common objective. Internet searches, cloud computing and artificial

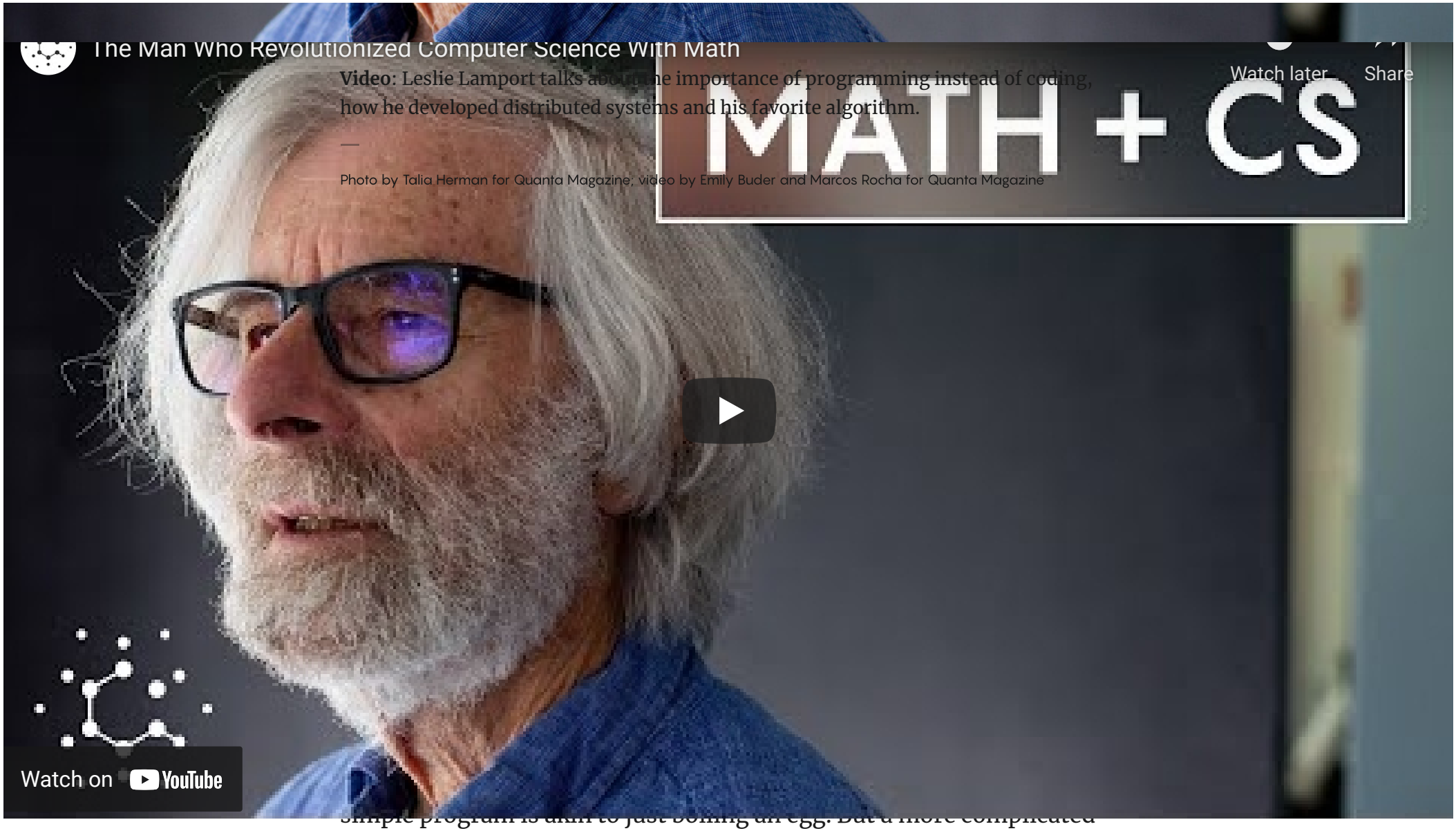
intelligence all involve orchestrating legions of powerful computing machines to work together. Of course, this kind of coordination opens you up to more problems.

“A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable,” Lamport once said.

Among the biggest sources of problems are “concurrent systems,” where multiple computing operations happen during overlapping slices of time, leading to ambiguity: Which computer’s clock is the right one? In a seminal 1978 paper, Lamport introduced the notion of “causality” to solve this issue, using an insight from special relativity. Two observers may disagree on the order of events, but if one event causes another, that eliminates the ambiguity. And sending or receiving a message can establish causality among multiple processes. Logical clocks — now also called Lamport clocks — provided a standard way to reason about concurrent systems.

With this tool in hand, computer scientists next wondered how they could systematically make these connected computers even bigger, without adding bugs. Lamport came up with an elegant solution:

Paxos, a “consensus algorithm” that allows multiple computers to execute complex tasks. Without Paxos and its family of algorithms, modern computing could not exist.



Video: Leslie Lamport talks about the importance of programming instead of coding, how he developed distributed systems and his favorite algorithm.

—

Photo by Talia Herman for Quanta Magazine; video by Emily Buder and Marcos Rocha for Quanta Magazine

simple program is akin to just boiling an egg. But a more complicated task with higher stakes — the coding equivalent of a nine-course banquet — requires more precision. You need to prepare each

component of each dish, combine them in a precise way, then serve them to every guest in the correct order. This requires exact recipes and instructions, written in unambiguous and succinct language, but descriptions written in English prose could leave room for misinterpretation. TLA+ employs the precise language of mathematics to prevent bugs and avoid design flaws.

Using your recipe, or specification, as an input, a program called a model checker will check whether the recipe makes sense and works as intended, producing a dish the way the chef wants it. Lamport laments how programmers often cobble together a system before writing a proper specification, whereas chefs would never cater a banquet without first knowing that their recipes will work.

Quanta spoke with Lamport about his work on distributed systems, what's wrong with computer science education, and how using TLA+ can help programmers build better systems. The interview has been condensed and edited for clarity.

Let's start with Paxos, since it's such an influential algorithm. What made you start working on it in the first place?

People were building a system with some code, and I had the hunch that what their code was trying to accomplish was impossible. So I decided to try to prove it, and instead came up with an algorithm that the people should have been using for their system.

What was wrong with their original algorithm?

Well, they didn't have an algorithm, just a bunch of code. Very few programmers think in terms of algorithms. When trying to write a concurrent system, if you just code it without having algorithms, there's no way that your program is not going to be full of bugs.

Lamport visits the Computer History Museum in Mountain View, Calif.

—

Talia Herman for Quanta Magazine

The paper that introduced Paxos wasn't very widely read at first. Why was that?

What made it impossible for people to read the paper was that I like explaining things with stories, and I made up names for characters in sort of pseudo-Greek letters. For example, in the paper there was a cheese inspector named Γωυδα. Having grown up as a mathematician, where Greek letters were used all over the place, I was just unaware that nonmathematicians get completely freaked out by those letters. Apparently, the readers couldn't deal with it, and it caused that paper not to be read as it should have been.

So that didn't work as well at first. Although in the long run it did, because people call this family of consensus algorithms Paxos instead of "viewstamped replication," which was another name for the same algorithm from [the computer scientist] Barbara Liskov.

After working on distributed systems for so many years, what got you into TLA+?

In the 1970s, when people were reasoning about programs, they were proving properties of the program itself stated in terms of programming languages. Then people realized that they should really be stating what the program is supposed to accomplish first — the program's behaviors.

In the early 1980s, I realized that one practical method of writing these higher-level specifications for concurrent systems was writing them as abstract algorithms. With TLA+, I was able to express them mathematically in a completely rigorous fashion. And everything clicked. What that involves is basically not trying to write algorithms in a programming language: If you really want to do things right, you need to write your algorithm in the terms of mathematics.

“The importance of thinking and writing before you code needs to be taught in undergraduate computer science courses and it’s not,” Lamport said.

Talia Herman for Quanta Magazine

You’ve said, “If you’re thinking without writing, you only think you’re thinking.” Is that where model checking comes in?

Model checking is a method for exhaustively testing all executions of a small model of the system. It just shows the correctness of the model, not of the algorithm. While model checking tests for correctness, coding just produces code. It doesn’t test anything. Before there was model checking, the only way to be sure that your algorithm worked was to write a proof.

In practice, model checking checks all executions of a small instance of the algorithm. And if you’re lucky, you can check large enough instances that it gives you enough confidence in the algorithm. But the proof can prove its correctness for a system of any size and for any use of the algorithm.

It sounds like model checking is related to another method of program verification: interactive theorem proving using tools such as Coq. How are they different?

Coq was designed to do real mathematics and to be able to capture the reasoning that mathematicians do. It's what Georges Gonthier used to prove the four-color theorem, for example. A machine-checked proof of a mathematical statement shows that the statement is almost certainly true.

TLA+ is designed not for mathematicians but for engineers who want to prove the properties of their systems. In the 1990s, after having spent about 15 years writing proofs of concurrent algorithms, I learned what you needed to do in order to prove the correctness of a concurrent algorithm. TLA was the logic that allowed it to be all completely formal. And TLA+ is the complete language based on that.

Lamport won the A.M. Turing Award in 2013 for his work on computer coordination, a field known as distributed systems. His Paxos algorithm is now an industry standard.

Talia Herman for Quanta Magazine

Specification languages like TLA+ aren't used very widely in industry, right? Why do you think that is?

Well, I'm doing what I can. But basically, programmers and many (if not most) computer scientists are terrified by math. So that's a tough sell.

Secondly, every project has to be done in a rush. There's an old saying, "There's never time to do it right. There's always time to do it over." Because TLA+ involves upfront effort, you're adding a new step in the development process, and that's also a hard sell.

Is it always worth that upfront effort?

True, most of the code written by programmers across the world doesn't require very precise statements about what it's supposed to do. But there are things that are important and need to be correct.

When people build a chip, they want that chip to work right. When people build a cloud infrastructure, they don't want bugs that will lose people's data. For the kind of application where precision is important, you need to be very rigorous. And you need something like TLA+,

especially if there's concurrency involved, which there usually is in these systems.

The specification language TLA+, developed by Lamport over the past few decades, allows engineers to describe a program's goals in a precise, mathematical way.

—

Talia Herman for Quanta Magazine

Are programmers biased in terms of spending more time writing code than they do thinking about it?

Yes, the importance of thinking and writing before you code needs to be taught in undergraduate computer science courses and it's not. And the reason is that there's no communication between the people who teach programming and the people who teach program verification.

From what I've seen, the fault lies on both sides of that divide. The people who teach programming don't know the verification that they need to know. The people who are teaching verification don't understand how it should be applied and used in practice.

Until that divide is bridged, TLA+ is not going to find a large number of users. I hope I could at least get the people who teach concurrent programming to understand that they need it. Then maybe there's some hope.

I get the sense that you aren't too happy with computer science education these days. Is it because it doesn't put enough emphasis on mathematics?

On mathematical thinking, yeah.

RELATED:

1. The Computer Scientist Who Can't Stop Telling Stories
2. The Architect of Modern Algorithms
3. How Close Are Computers to Automating Mathematical Reasoning?
4. Building the Mathematical Library of the Future

How would you structure an undergraduate curriculum, then?

I'm not an educator, so I don't know how to teach it to them. But I know what people should have learned. They shouldn't be afraid of math. It's just simple math that they've probably taken a course in, but they don't know how to use it. They don't know what good it is. They learn enough to pass the exam and then they forget about it.

Mathematicians often say they see beauty in math. You started out in that field, so do you see beauty in algorithms?

I don't think in terms of aesthetics. I probably have the kinds of feelings that other people do, but I just use different words to express

them. Being beautiful is not something I would say about an algorithm. But simplicity is something that I value highly.

One last thing, about another side project of yours with a sizable impact: LaTeX. I'd like to finally clear something up with the creator. Is it pronounced LAH-tekh or LAY-tekh?

Any way you want. I don't advise spending very much time thinking about it.

Sheon Han

Contributing Writer

May 17, 2022

computer science | computers | proofs | Q&A | All topics →

The Quanta Newsletter

Get highlights of the most important news delivered to your email inbox

Email address | Subscribe

Recent newsletters

Also in Computer Science

ALGORITHMS

Powerful 'Machine Scientists' Distill the Laws of Physics From Raw Data

By CHARLIE WOOD | MAY 10, 2022 | 13 |

CRYPTOGRAPHY

Which Computational Universe Do We Live In?

By ERICA KLARREICH | APRIL 18, 2022 | 20 |

NEURAL NETWORKS

Researchers Gain New Understanding From Simple AI

By MORDECHAI RORVIG | APRIL 14, 2022 | 2 |

Comment on this article

Quanta Magazine moderates comments to facilitate an informed, substantive, civil conversation. Abusive, profane, self-promotional, misleading, incoherent or off-topic comments will be rejected. Moderators are staffed during regular business hours (New York time) and can only accept comments written in English.

ALSO ON **QUANTA MAGAZINE**

2 days ago • 9 comments

By resolving a paradox

7 days ago • 4 comments

Finding out whether a

14 days ago • 12 comr

Einstein's descripti

Show comments