DOI:10.1145/3107924

Dominic Horsman, Vivien Kendon, and Susan Stepney

Viewpoint
The Natural Science of Computing

As unconventional computing comes of age, we believe a revolution is needed in our view of computer science.

TECHNOLOGY CHANGES SCIENCE. In 2016, the scientific community thrilled to news that the LIGO collaboration had detected gravitational waves for the first time. LIGO is the latest in a long line of revolutionary technologies in astronomy, from the ability to 'see' the universe from radio waves to gamma rays, or from detecting cosmic rays and neutrinos (the Laser Interferometer Gravitational-Wave Observatory—LIGO—is an NSF-supported collaborative effort by the U.S National Science Foundation and is operated by Caltech and MIT). Each time a new technology is deployed, it can open up a new window on the cosmos, and major new theoretical developments can follow rapidly. These, in turn, can inform future technologies. This interplay of technological and fundamental theoretical advance is replicated across all the natural sciences—which include, we argue, computer science. Some early computing models were developed as abstract models of existing physical computing systems. Most famously, for the Turing Machine these were human 'computers' performing calculations. Now, as novel computing devices—from quantum computers to DNA processors, and even vast networks of human 'social machines'—reach a critical stage of development, they reveal how computing technologies can drive the expansion of theoretical tools and models of computing. With all due respect to Dijkstra, we argue that computer science is as much about computers as astronomy is about telescopes.

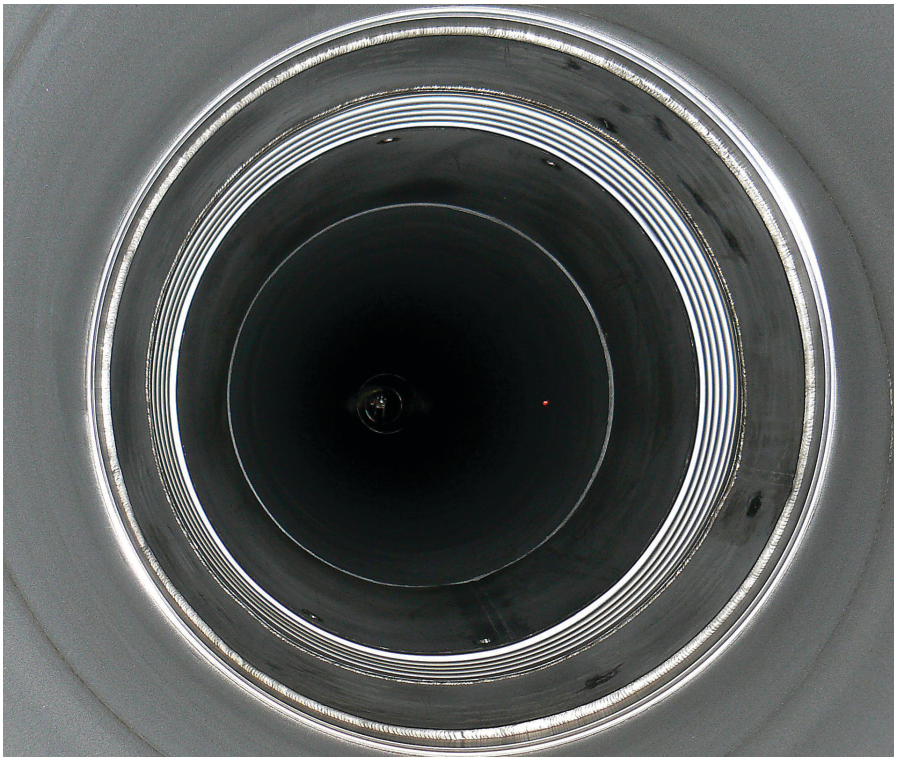


Image of the Laser Interferometer Gravitational-Wave Observatory 40m beam tube.

PHOTO BY TOBIN FRICKE/FLICKR

Non-standard and unconventional computing technologies have come to prominence as Moore's Law, that previously relentless increase in computing power, runs out. While techniques such as multicore and parallel processing allow for some gains without further increase of transistor density, there is a growing consensus that the next big step will come from technologies outside the framework of silicon hardware and binary logic. Quantum computing is now being developed on an international scale, with active research and use from Google and NASA as well as numerous universities and national laboratories, and a proposed €1 billion quantum technologies flagship from the European Commission. Biological computing is also being developed, from data encoding and pro-

cessing in DNA molecules, to neuro-silicon hybrid devices and bio-inspired neural networks, to harnessing the behavior of slime molds. The huge advance of the internet has enabled 'social machines'—Galaxy Zoo, protein FoldIt, Wikipedia, innumerable citizen science tools—all working by networking humans and computers, to perform computations not accessible on current silicon-based technology alone.

What all these devices, from the speculative to the everyday, share is that they currently lie beyond the reach of conventional computer science. Standard silicon-based technology is built on a toolkit of theoretical models and techniques, from lambda calculi through to programming, compilation, and verification. These tools seem to be largely inaccessible to the new technologies. How do you program a slime mold? What is the assembly language of protein folding? How do you compile for a human in a social machine? New technologies may be one or more of stochastic, continuous time, continuous space, sloppy, asynchronous, temperature dependent, sub-symbolic, evolving systems, with computationally complex encodings and decodings, and one shot construction-and-execution.

Without the ability to define and characterize how and when computing is happening in these systems, and then to import or develop the full suite of theoretical tools of computer science, we claim that the information-processing capabilities of these devices will remain underexploited. We believe we need an extended computer science that will enable us to treat these systems with theoretical and practical rigor to unlock their potential, and also to enable us to combine them with existing technology to make scalable and hybrid devices.

Computer science has historically been conceived and developed around abstract Turing Machines and equivalent calculi. This discrete, symbolic logical, deterministic underlying model is realized equivalently, but differently, in one specific technology, the von Neumann stored program architecture. This technology has proved so successful, and is now so ubiquitous, that other models of computing have tended to be ignored; one example is Shannon's largely forgotten GPAC computational model, based on the technology of differential analysers. As a consequence of using only a single model, standard approaches to computing abstract away the physical implementation, leaving a theoretical computer science that is frequently viewed *as* a branch of mathematics, rather than as a physical science that is expressed *in* mathematical language. With little connection to actual physical devices, this theoretical framework can be at a loss when faced with nonstandard computing systems. Often the response is to impose top-down a standard bit-and-logic-gate framework, in the belief that this is *the* way to compute. The delicate systems in a quantum computer, for instance, can be forced to act like standard bits obeying classical logic. However, these devices gain their real power when allowed to act as natively 'quantum bits,' or qubits, with their own quantum logic gates. It is as inefficient (or simply impossible) to impose the standard computing framework on many nonstandard systems as it would be to use a sophisticated optical telescope to detect cosmic neutrinos. We do not believe that we can unlock the true potential of unconventional systems by forcing them into the mold of standard computing models.

It is inefficient (or simply impossible) to impose the standard computing framework on many nonstandard systems.

While traditionally computer science tends to view itself as a branch of mathematics, the field of unconventional computing has tended to go too far the other way, seeing computing merely as an outgrowth of physics, or chemistry, or biology. Arguments around computing power often overfocus on the physical *theory* of the device, rather than what can be implemented in the physical device itself. Neural nets, for example, can be modeled using real-valued activation functions, and arguments have been made that these networks are in actuality computing those real values to arbitrary precision, and hence far outperforming the capabilities of standard computers. In practice, however, such purely abstract infinite real-valued precision is completely outside the physical capabilities of the device: it can neither be observed, nor exploited.

Computing theory should not be imposed top-down without taking into account the physical theory of the device: computer science is not mathematics. The computing ability of the system is not always identical with the computing capability of the physical device theory: computer science is not physics. What is it, then? We believe that it has features of both, consisting in the complex interplay of mathematics and physical theory through a crucial relation: representation.

Understanding computers can be seen as the key to understanding computer science. A computer crosses the boundary between the abstract domain of logic/computation, and the physical realm of semiconductors or quantum ions or biological molecules; and it does so in a way that we can precisely characterize. Consider part (a) of the the figure here, in which a 'compute cycle' starts with an abstract problem, such as adding two numbers, or finding prime factors, or calculating a shortest path. Usually expressed in some high-level language, this is then *encoded* into the computer's native language. This encoding is still in essence an abstract process: the description of the computation has been transformed from one language to another. Now the actual computer is brought in, and the native language input is *instantiated* in the target physical device. The device is configured, and the physical processes of computing initialized. Then the computer runs, as a *physical* process with a physical output in the final state of the computer. To find the output of the computation, we ask to what abstract state the physical one corresponds: which state of the program is *represented* by the physical state of the computer? This is then (abstractly) *decoded* from

the abstracted output to a language to answer the original problem. The computer has output the solution.

If a computer is a good one, and running without errors, the aim of the compute cycle is to parallel the physical and abstract behaviours. The solution is an abstract answer to an abstract question; were it possible to "run" the program entirely abstractly, then the solution could be found without including any physical device in the cycle, be that an engineered computer, or a pencil-and-paper based human emulation. Computers are used as a physical proxy for this abstract mapping. A good computer is engineered such that the result of letting the physical dynamics run will parallel the abstract behavior of the program. A computer is a device that manipulates the physical instantiation of abstract concepts, in order to solve problems. It is not identical with a computation: computation is abstract, a computer is physical, and they relate through (nontrivial) representation and instantiation.

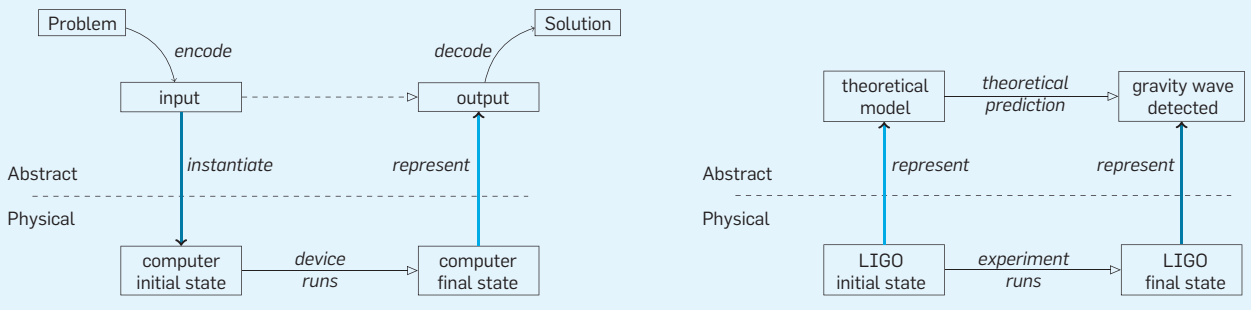
This centrality of representation is the core of a new formalism developed by the authors: Abstraction/Representation Theory (AR theory). With diagrams such as part (a) of the figure here, and an associated algebraic-like structure, AR theory is a toolkit for the foundations of computer science, and beyond. The complex interplay of mathematics, physical theory, and representation is not confined to the field of computing. It also drives the mechanism of experimental testing of abstract theories throughout the natural sciences. We can, for instance, give a diagram for the relation between astronomy and telescopes, with crucial similarities and differences to computing. Part (b) of the figure shows how theory and experiment relate in the LIGO experiment. Again abstract and physical are parallel, but now the process of running the experiment starts with the physical apparatus, rather than with an encoding of an abstract computational problem. There is an abstract representation of the apparatus in the theory of gravitational waves: that it can detect them. Also in the abstract realm there is a theoretical prediction for how the apparatus will behave if such waves indeed exist. If the experiment is successful, as with LIGO, then the theory and the abstract interpretation of the physical outcome coincide up to some error margin ε. If a theory is sufficiently good, the physical system can be removed altogether: abstract theory can be used to predict physical behavior.

Looking at these two diagrams, we uncover a deep truth. Just as a mathematical theory allows us to predict physical behavior, in a computer the physical behavior of a device is used to 'predict' the result of an abstract computation. Computing and natural science are fundamentally linked; the link is technology. Notice the direction of the arrows of representation in the two diagrams. In an experiment, they go only one way, upward: this is the *representation* of physical systems by an abstract model. In computing there is another type of relation: *instantiation* of abstract theory in physical systems. Instantiation is more complex than modelling, and requires *engineering* to construct a system that, when modeled, gives the abstract specification to be instantiated; this engineering in turn requires a sufficiently good scientific understanding of the system's properties. Not all abstract systems that can be imagined denote something in the physical world ("the present king of France"), or can be physically instantiated (faster-than-light travel).

Just like a telescope, a computer is a highly engineered device. LIGO went through many years of testing of its various components before scientists were happy that it would function as a gravitational wave detector. With the tests complete, it can now be used as a telescope to observe the universe in terms of those ripples. Similarly, computers require engineering before they can be used for computation: we need to be confident that their physical behavior parallels that of the abstract program so that the device can be used to predict its outcome (there can be engineering bugs in hardware). Computers start with computer *science*: with experiments on novel substrates and with new ways of performing computing. Only once that cycle is complete, and we know enough about how the system behaves, can a new device be engineered to instantiate a computation.

What does AR theory mean for our understanding of computer science? We claim that we now have a way to understand that computational logic arises from the physical structure of a potential computing substrate, and that it may vary widely across different classes of substrate. Computer science, in addition to its theoretical component, covers both the experimentation

Computers and telescopes: The interplay of abstract theory/programming and physical devices in (a) computing and (b) physical sciences (given here by the LIGO experiment).



and engineering phases of computing, as well as the eventual use in deployment as a computer. This understanding tells us to use an experimental and engineering process when developing new formal models and methods of computer sciences for our new devices, paralleling the process of developing new models and instruments to tackle new phenomena in rest of the natural sciences. A computational logic for a system arises, but we then abstract away from the specific device to a formal model of it. Programming these new devices is then a matter of looking for a natural internal process logic of the system, as opposed to forcing a one-size-fits-all model of computation onto some candidate computing system. Rather than looking to impose top-down the machinery of standard logic gates, we should look at the natural behaviour of the system and what 'gates' or subroutines or problem-solving it is intrinsically good at. By extracting an intrinsic computational logic of their physical components we can harness the true potential of unconventional computers.

Using our physical understanding of a substrate to inform a computational logic does not mean that such a logic is the only one possible. Just as a quantum computer can run as either quantum or classical, other nonstandard systems may be capable of supporting multiple computational models. This again is found throughout the natural sciences: for example, in physics a particular system might be modelled as a continuous fluid, or as a collection of discrete particles. With different potential computational representations of a system under investigation, the key is to extract out the ones that do something useful and novel and better than other substrates—and then use that computational theory to engineer our next generation of computers.

We can then go further. With an abstract computational language that describes the native operation of unconventional devices, we would then have a logical language in which to describe the physical systems themselves, even outside a specifically computational device. Computer science could then provide high-level logical process languages for physics, chemistry, and biology—and even, with the interactions of social machines, for networks of human beings. We believe this could be of immediate practical importance to scientists in those areas, enabling them to describe high-level functioning of complex systems, and to find new and unforeseen connections between disparate systems and scenarios. These process languages could be as revolutionary for the physical sciences as for computer science.

Computers have come a long way since the days of valves and punched cards. Now computer science itself is branching off in new directions with the development of unconventional computing technologies. As the domain of computer science grows, as one computational model no longer fits all, its true nature is being revealed. Just like astronomy, computer science could describe physical systems in abstract language with predictive power, and thereby drive forward the dual interplay of technology and theoretical advancement. New computers could inform new computational theories, and those theories could then help us understand the physical world around us. Such a computer science would indeed be a natural science. C

Further Reading

Copeland, J. et al. Time to reinspect the foundations? *Commun. ACM 59*, 11 (Nov. 2016), 34–36.

Horsman, C. et al. When does a physical system compute? In *Proceedings of the Royal Society of London*, 470:20140182, 2014.

Horsman, D.C. Abstraction/Representation Theory for heterotic physical computing. In *Philosophical Transactions of the Royal Society*, 373:20140224, 2015.

Horsman, D.C. Abstraction and representation in living organisms: When does a biological system compute? In G. Dodig-Crnkovic and R. Giovagnoli, Eds. *Representation and Reality in Humans, Animals, and Machines*. Springer, 2017.

Dominic Horsman (dominic.horsman@durham.ac.uk) is a Postdoctoral Research Associate at the University of Durham, U.K.

Vivien Kendon (viv.kendon@durham.ac.uk) is a Reader in the Department of Physics at the University of Durham, U.K.

Susan Stepney (susan.stepney@york.ac.uk) is Professor of Computer Science in the Department of Computer Science, University of York, U.K.