Ubiquity, an ACM publication
April, 2011

Ubiquity Symposium

What Have We Said About Computation?

Closing Statement

by Peter J. Denning

Editor's Introduction

The "computation" symposium presents the reflections of thinkers from many sectors of computing on the fundamental question in the background of everything we do as computing professionals. While many of us have too many immediate tasks to allow us time for our own deep reflection, we do appreciate when others have done this for us. Peter Freeman points out, by analogy, that as citizens of democracies we do not spend a lot of time reflecting on the question, "What is a democracy," but from time to time we find it helpful to see what philosophers and political scientists are saying about the context in which we act as citizens.

Ubiquity Symposium

What Have We Said About Computation?

Closing Statement

by Peter J. Denning

This symposium has been a provocative learning experience for readers, authors, and editors. I would like to share some of what I have learned in working with the 14 authors. My general conclusions are grouped into three sections:

- Points of major agreement
- Questions not fully resolved
- Interactive systems

The first section lists the points in which everyone seems to be in agreement. In my opening statement, I expressed concerns that developments in non-terminating computation, analog computation, and natural computation may require rethinking the basic definitions of computation. The authors agree that these developments easily fit within existing understandings of computing. The second section highlights areas where some of the authors disagree among themselves. Further reflection will be required to resolve those issues. I'm sure some of them seem pretty esoteric! The third, and final, section focuses on a class of computational systems called interactive or reactive systems, which have been around for a long time but are fundamentally different from the computational systems envisioned by Turing.

Points of Major Agreement

Computation is a process. Every author distinguishes the machine or algorithm from the process that the machine or algorithm generates. Dennis Frailey makes the argument most directly and wonders if every process is also a computation. In analogies: the machine is a car, the desired outcome is the driver's destination, and the computation is the journey taken by the car and driver to the destination.

Computational model matters. A computational model is a specification of a method for data representation and a mechanism that transforms those representations toward a desired outcome. Here are some examples. A Turing machine is an abstract model of a sequential digital computer with finite, but unbounded, memory. A finite state machine is a model for bounded-memory process controllers. A Petri Net is a model for a network of asynchronous parallel tasks. String rewriting rules are a model for DNA translation. For analysis and design, we want models that reflect the domain closely and enable prediction. Part of the work of scientists and engineers is to discover or design new models that deal with new domains.

Many important computations are natural. Although our tradition attunes our thinking to machine-generated computations, there are now numerous examples of natural ones. Dave Bacon notes this for physics, Melanie Mitchell for biology, and Erol Gelenbe for other natural processes. Prominent examples of natural computation include DNA translation, brain processes, quantum information, and social network games.

Many important computations are non-terminating. The original definitions in the 1930s associated computation with terminating algorithms. This was understandable because the initial focus was on automatic calculation of mathematical functions; a non-terminating calculation would, by definition, not be able to calculate a number. In the 1960s, operating system designers invented the term "process" for a non-terminating computation. Many services of operating systems —such as network protocols, web servers, and login sessions— are designed to continue indefinitely. They cannot do their work if they terminate prematurely. Today, everyone seems to accept that some computations terminate and others do not. In his 1986 book *Finite and Infinite Games*, James Carse says: "A finite game is played for the purpose of winning, an infinite game for the purpose of continuing the play." Developers who focus on generating a result then stopping are designing a finite game; those who focus on keeping the process going indefinitely are designing an infinite game.

Many important computations are continuous. Our familiarity with the Turing model inclines us to think of computation as discrete. However, physical implementations of automatic computation use continuous signals. We discretize the states of the signals to improve the reliability of detection and interpretation in the face of imprecise measurement, roundoff errors, and noise. Discretized systems may be called "covertly continuous" because the continuous signals used to represent quantities are normally hidden. Some systems are overtly continuous, such as mathematical models of heat flow, magnetic pole phase change, or galaxy collision. Joe Traub details the methods of analysis used to predict running times of these models on digital computers. Even though the analytic methods may differ, continuous systems do not have more computing power than digital systems.

Computational thinking can be defined. The term "algorithmic thinking" was used in the 1950s to describe an essential difference between problem solving in computing and in other fields. The term "computational thinking" was used in the 1980s to describe the new way of doing science enabled by supercomputers. These terms stayed quietly in the background until 2006, when Jeanette Wing popularized them as part of what everyone should know about computing. Her account did not precisely define computational thinking and left many educators asking "what is computational thinking?" Several authors have provided their one-sentence answers to this. I said: "Computational thinking is an approach to problem solving that represents the problem as an information process relative to a computational model (which may have to be invented or discovered) and seeks an algorithmic solution." Al Aho offered the shortest: "We consider computational thinking to be the thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms." We should avoid the trap of concluding that computational thinking defines computing. For example, computational thinking is not the same as "computational doing," which is the actual work of computing professionals as they make computation real and effective in their worlds.

Questions Not Fully Resolved

Is the Turing machine the only acceptable reference model? The Turing machine became the reference model for computation in the 1930s. The Church-Turing thesis of the same era held that any effective method for computing a function could be realized by an appropriate Turing machine. Since then, most new proposals for computational models have been compared to the Turing machine. No one has found a model that can compute a function that a Turing machine cannot. For this reason, by the 1960s, the Turing model was broadly accepted as a reference model. Lance Fortnow takes a strong position for the Turing model, which he believes models the process by which numbers are computed, and is fundamentally the same whether or not the machine terminates. John Conery says that, as in the Turing model, computation is the series of state-transitions in a process that represent states and inputs symbolically. He relaxes the Turing model a bit by allowing state transitions not controlled by algorithms, as in natural or biological computation. However, as the importance of computation has flourished in many fields, the ability of the Turing machine model to predict running times of computations has been mixed. It is often found that computational models closely matched to the problems in a domain are more accurate predictors. Joe Traub reports this is true for computational science, which deals with continuous problems, while Jeff Buzen reports the same for models that predict throughput and response time of computing systems under workload uncertainty. Some of the authors argue that the Turing machine is a fine reference model; others argue that alternative models (even if Turing equivalent) are better predictors for performance in specific domains.

Is information observable? Most of the authors define computation as some sort of transformation of information, but do not provide a clear definition of information. Paul Rosenbloom goes to some length to define information and its relationship to computation. Dave Bacon says information is part of the quantum structure of the universe, but no one has directly observed it. Ruzena Bajcsy says that some information is manifested in physical measurements such as voltages, sound waves, or electron spins, and other information is synthesized from human thoughts such as mathematical models; we come to know the latter when they are expressed in symbols and language conveyed by the former. I admit to considerable confusion myself on this. The formal definitions of data (objective symbols) and information (subjective meaning) do not help me design computers and algorithms. I think it is possible to engineer computers and analyze their performance and limitations without having to resolve the meaning question. Still, what information is remains an open question.

Is all computation physical? The term "representation" means a pattern of symbols standing for something. The "standing for" part is a statement of social convention: We have come to collective agreements on what various patterns mean to us. We usually think of the representation as something recorded in a medium where others can inspect it, and meaning as the mental state evoked by a representation. Yet it is possible to define representations as abstract languages and algorithms as a means to impose orderings on the strings in languages, and then answer questions about the limits of machines that recognize the languages or their running time when analyzing strings. Paul Rosenbloom argues that some computation is purely abstract. On the other hand, Dave Bacon says that computation is a construction of nature to overcome uncertainties in unreliable information processes. This question remains unresolved.

Interactive Systems

The computing theory community has defined "reactive systems" as a class of computational systems that model interactive computations. These systems receive ongoing inputs and provide ongoing outputs. Their purpose is to maintain interactions indefinitely, rather than to compute functions and stop. They are a form of Carse's infinite game mentioned earlier.

Reactive (interactive) systems are not new; they have been studied since the 1950s by operating system engineers and later by network and database system engineers. Those engineers, however, were more interested in the models as means to design well-behaved and predictable systems, not as models of computation.

The field of operating systems has been a rich source of computational models of reactive systems. Examples include Petri Nets (1939), cooperating sequential processes (Dijkstra 1968), schemata (Karp-Miller 1969), partially ordered task sets (Coffman-Denning 1973), communicating processes (Hoare 1978), and operational queueing networks (Buzen-Denning 1977). These models have been used to analyze race conditions, process creation and deletion, asynchronous communication, determinacy, deadlock, performance, and fault resistance. It should be noted that Turing machine models address none of these phenomena.

In this symposium, Peter Wegner argues that interactive systems are fundamentally different from Turing models, and Al Aho agrees. A Turing machine begins with a finite input and aims to produce a finite output. An interactive system, on the other hand, has an ongoing input that continues indefinitely and produces an ongoing output that also continues indefinitely. All the input to a Turing machine is presented before it starts, but interactive systems constantly receive new inputs after they start, and they produce outputs before all inputs are received. Interactive systems may have many asynchronous channels of input and output. The input-output relationship of an interactive system is not a function, as it is in a Turing machine. Even when an interactive system's outputs can be functionally related to its inputs, many inputs may be non-computational—such as humans inputting their decisions. These differences lead some authors to doubt whether Turing machines can simulate interactive systems or, in other words, whether interactive systems are equivalent to Turing machines. On top of all this, an interactive system may dynamically change configuration while it is computing—for example, adding or deleting nodes or links in a network.

In designing an interactive system such as the Internet, the designer is interested in issues such as capacity, throughput, and response time—none of which can be evaluated with a Turing machine. In this symposium, Jeff Buzen provides a model of reactive/interactive systems for evaluating such questions when the workloads that drive the system are uncertain, while retaining the underlying deterministic system of computing machines.

Conclusions

I was struck by the clarity that comes when we make explicit the computational model we are working with. The Turing machine model is not the most convenient for many domains even though theoretically it is equivalent with every convenient model in its ability to define functions. In solving real problems, we work with computational models (or invent new ones) that are convenient and appropriate for the domain. There is an emerging consensus that reactive systems are models of interactive computation and are different from Turing modeled systems.

I noted a considerable amount of agreement on what issues are settled and what issues remain open. When I took polls about what is computer science and computing in the mid 1980s, I was struck by the variation of opinion among computing educators on exactly what constitutes computation and therefore what topics we should teach our students. Today's consensus replaces yesterday's diversity.

Finally, I am amazed at the depth of understanding about computation that we collectively have achieved since the 1960s. We have made remarkable progress toward maturity as a field.

About the Author

Peter J. Denning is director of the Cebrowski Institute for innovation and information superiority at the Naval Postgraduate School in Monterey, California, and is a past president of the ACM. He is currently the editor in chief of *Ubiquity*.

DOI: 10.1145/ 1967045.1967046