Ubiquity, an ACM publication
January, 2011

Ubiquity Symposium

What is Computation?

Computation and Computational Thinking

by Alfred V. Aho,
Department of Computer Science, Columbia University

Editor's Introduction

In this ninth piece to the Ubiquity symposium discussing 'What is computation?' Alfred V. Aho shares his views about the importance of computational thinking in answering the question.

Peter J. Denning
Editor

Ubiquity Symposium

What is Computation?

Computation and Computational Thinking

by Alfred V. Aho,
Department of Computer Science, Columbia University

The Need for Clear Definitions

In any scientific discipline there are many reasons to use terms that have precise definitions. Understanding the terminology of a discipline is essential to learning a subject and precise terminology enables us to communicate ideas clearly with other people. In computer science the problem is even more acute: we need to construct software and hardware components that must smoothly interoperate across interfaces with clients and other components in distributed systems. The definitions of these interfaces need to be precisely specified for interoperability and good systems performance.

Using the term "computation" without qualification often generates a lot of confusion. Part of the problem is that the nature of systems exhibiting computational behavior is varied and the term computation means different things to different people depending on the kinds of computational systems they are studying and the kinds of problems they are investigating. Since computation refers to a process that is defined in terms of some underlying model of computation, we would achieve clearer communication if we made clear what the underlying model is.

Rather than talking about a vague notion of “computation,” my suggestion is to use the term in conjunction with a well-defined model of computation whose semantics is clear and which matches the problem being investigated. Computer science already has a number of useful clearly defined models of computation whose behaviors and capabilities are well understood. We should use such models as part of any definition of the term computation. However, for new domains of investigation where there are no appropriate models it may be necessary to invent new formalisms to represent the systems under study.

Computational Thinking

We consider computational thinking to be the thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms. An important part of this process is finding appropriate models of computation with which to formulate the problem and derive its solutions. A familiar example would be the use of finite automata to solve string pattern matching problems. A less familiar example might be the quantum circuits and order finding formulation that Peter Schor used to devise an integer-factoring algorithm that runs in polynomial time on a quantum computer. Associated with the basic models of computation in computer science is a wealth of well-known algorithm-design and problem-solving techniques that can be used to solve common problems arising in computing.

However, as the computer systems we wish to build become more complex and as we apply computer science abstractions to new problem domains, we discover that we do not always have the appropriate models to devise solutions. In these cases, computational thinking becomes a research activity that includes inventing appropriate new models of computation.

Corrado Priami and his colleagues at the Centre for Computational and Systems Biology in Trento, Italy have been using process calculi as a model of computation to create programming languages to simulate biological processes. Priami states "the basic feature of computational thinking is abstraction of reality in such a way that the neglected details in the model make it executable by a machine." [Priami, 2007]

As we shall see, finding or devising appropriate models of computation to formulate problems is a central and often nontrivial part of computational thinking.

Forces at Play

In the last half century, what we think of as a computational system has expanded dramatically. In the earliest days of computing, a computer was an isolated machine with limited memory to which programs were submitted one at a time to be compiled and run. Today, in the Internet era, we have networks consisting of millions of interconnected computers and as we move into cloud computing, many foresee a global computing environment with billions of clients having universal on-demand access to computing services and data hosted in gigantic data centers located around the planet. Anything from a PC or a phone or a TV or a sensor can be a client and a data center may consist of hundreds of thousands of servers. Needless to say, the models for studying such a universally accessible, complex, highly concurrent distributed system are very different from the ones for a single isolated computer.

Another force at play is that because of heat dissipation considerations the architecture of computers is changing. An ordinary PC today has many different computing elements such as multicore chips and graphics processing units, and an exascale supercomputer by the end of this decade is expected to be a giant parallel machine with up to a million nodes each with possibly a thousand processors. Our understanding of how to write efficient programs for these machines is limited. Good models of parallel computation and parallel algorithm design techniques are a vital open research area for effective parallel computing.

In addition, there is increasing interest in applying computation to studying virtually all areas of human endeavor. One fascinating example is simulating the highly parallel biological processes found in human cells and organs for the purposes of understanding disease and drug design. Good computational models for biological processes are still in their infancy. And it is not clear we will ever be able to find a computational model for the human brain that would account for emergent phenomena such as consciousness or intelligence.

The Theory of Computation

The theory of computation has been and still is one of the core areas of computer science. It explores the fundamental capabilities and limitations of models of computation. A model of computation is a mathematical abstraction of a computing system. The most important model of sequential computation studied in computer science is the Turing machine, first proposed by Alan Turing in 1936. Let us briefly review the definition of a Turing machine to appreciate the detail necessary to understand even this familiar model of computation.

We can think of a Turing machine as a finite-state control attached to a tape head that can read and write symbols on the squares of a semi-infinite tape. Initially, a finite string of length *n* representing the input is in the leftmost *n* squares of the tape. An infinite sequence of blanks follows the input string. The tape head is reading the symbol in the leftmost square and the finite control is in a predefined initial state.

The Turing machine then makes a sequence of moves. In a move it reads the symbol on the tape under the tape head and consults a transition table in the finite-state control which specifies a symbol to be overprinted on the square under the tape head, a direction the tape head is to move (one square to the left or right), and a state to enter next. If the Turing machine enters an accepting halting state (one with no next move), the string of nonblank symbols remaining on the input tape at that point in time is its output.

Mathematically, a Turing machine consists of seven components: a finite set of states; a finite input alphabet (not containing the blank); a finite tape alphabet (which includes the input alphabet

and the blank); a transition function that maps a state and a tape symbol into a state, tape symbol, and direction (left or right); a start state; an accept state from which there are no further moves; and a reject state from which there are no further moves.

We can characterize the configuration of a Turing machine at a given moment in time by three quantities:

(1) the state of the finite-state control,
(2) the string of nonblank symbols on the tape, and
(3) the location of the input head on the tape.

A computation of a Turing machine on an input w is a sequence of configurations the machine

can go through starting from the initial configuration with w on the tape and terminating (if the computation terminates) in a halting configuration. We say a function *f* from strings to strings is computable if there is some Turing machine *M* that given any input string *w* always halts in the accepting state with just *f*(*w*) on its tape. We say that *M* computes *f*.

The Turing machine provides a precise definition for the term algorithm: an algorithm for a function *f* is just a Turing machine that computes *f*.

There are scores of models of computation that are equivalent to Turing machines in the sense that these models compute exactly the same set of functions that Turing machines can compute. Among these Turing-complete models of computation are multitape Turing machines, lambda-calculus, random access machines, production systems, cellular automata, and all general-purpose programming languages.

The reason there are so many different models of computation equivalent to Turing machines is that we rarely want to implement an algorithm as a Turing machine program; we would like to use a computational notation such as a programming language that is easy to write and easy to understand. But no matter what notation we choose, the famous Church-Turing thesis hypothesizes that any function that can be computed can be computed by a Turing machine.

Note that if there is one algorithm to compute a function *f*, then there is an infinite number. Much of computer science is devoted to finding efficient algorithms to compute a given function.

For clarity, we should point out that we have defined a computation as a sequence of configurations a Turing machine can go through on a given input. This sequence could be infinite if the machine does not halt or one of a number of possible sequences in case the machine is nondeterministic.

The reason we went through this explanation is to point out how much detail is involved in precisely defining the term computation for the Turing machine, one of the simplest models of computation. It is not surprising, then, as we move to more complex models, the amount of effort needed to precisely formulate computation in terms of those models grows substantially.

Concurrent Models

Many real-world computational systems compute more than just a single function—the world has moved to interactive computing [Goldin, Smolka, Wegner, 2006]. The term reactive system is used to describe a system that maintains an ongoing interaction with its environment. Examples of reactive systems include operating systems and embedded systems.

A distributed system is one that consists of autonomous computing systems that communicate with one another through some kind of network using message passing. Examples of distributed systems include telecommunications systems, the Internet, air-traffic control systems, and parallel computers. Many distributed systems are also reactive systems.

Perhaps the most intriguing examples of reactive distributed computing systems are biological systems such as cells and organisms. We could even consider the human brain to be a biological computing system. Formulation of appropriate models of computation for understanding biological processes is a formidable scientific challenge in the intersection of biology and computer science.

Distributed systems can exhibit behaviors such as deadlock, livelock, race conditions, and the like that cannot be usefully studied using a sequential model of computation. Moreover, solving problems such as determining the throughput, latency, and performance of a distributed system cannot be productively formulated with a single-thread model of computation. For these reasons, computer scientists have developed a number of models of concurrent computation which can be used to study these phenomena and to architect tools and components for building distributed systems.

There are many theoretical models for concurrent computation. One is the message-passing Actor model, consisting of computational entities called actors [Hewitt, Bishop, Steiger, 1973].

An actor can send and receive messages, make local decisions, create more actors, and fix the behavior to be used for the next message it receives. These actions may be executed in parallel and in no fixed order. The Actor model was devised to study the behavioral properties of parallel computing machines consisting of large numbers of independent processors communicating by passing messages through a network. Other well-studied models of concurrent computation include Petri nets and the process calculi such as pi-calculus and mu-calculus.

Many variants of computational models for distributed systems are being devised to study and understand the behaviors of biological systems. For example, Dematte, Priami, and Romanel [2008] describe a language called BlenX that is based on a process calculus called Beta-binders for modeling and simulating biological systems.

We do not have the space to describe these concurrent models in any detail. However, it is still an open research area to find practically useful concurrent models of computation that combine control and data for many areas of distributed computing.

Benefits of Models of Computation

In addition to aiding education and understanding, there are many practical benefits to having appropriate models of computation for the systems we are trying to build. In cloud computing, for example, there are still a host of poorly understood concerns for systems of this scale. We need to better understand the architectural tradeoffs needed to achieve the desired levels of reliability, performance, scalability and adaptivity in the services these systems are expected to provide. We do not have appropriate abstractions to describe these properties in such a way that they can be automatically mapped from a model of computation into an implementation (or the other way around).

In cloud computing, there are a host of research challenges for system developers and tool builders. As examples, we need programming languages, compilers, verification tools, defect detection tools, and service management tools that can scale to the huge number of clients and servers involved in the networks and data centers of the future. Cloud computing is one important area that can benefit from innovative computational thinking.

Conclusion

Mathematical abstractions called models of computation are at the heart of computation and computational thinking. Computation is a process that is defined in terms of an underlying model of computation and computational thinking is the thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms. Useful models of computation for solving problems arising in sequential computation can range from simple finite-state machines to Turing-complete models such as random access machines. Useful models of concurrent computation for solving problems arising in the design and analysis of complex distributed systems are still a subject of current research.

About the Author

Alfred V. Aho is Lawrence Gussman Professor in the Computer Science Department at Columbia University. He served as Chair of the department from 1995 to 1997, and in the spring of 2003.

Acknowledgments

The author would like to thank Peter Denning and Jeannette Wing for their thoughtful comments on the importance of computational thinking. The author is also grateful to Jim Larus for his insights into the problems confronting cloud computing and to Corrado Priami for many stimulating conversations on computational thinking in biology.

References

Dematte, L., Priami, C., and Romanel, A. The BlenX language, a tutorial. In M. Bernardo, P. Degano, and G. Zavattaro (Eds.): SFM 2008, LNCS 5016, pp. 313-365, Springer, 2008.

Denning, P. J. Beyond computational thinking. *Comm. ACM*, pp. 28-30, June 2009.

Goldin, D., Smolka, S., and Wegner, P. Interactive Computation: The New Paradigm, Springer, 2006.

Hewitt, C., Bishop, P., and Steiger, R. A universal modular ACTOR formalism for artificial intelligence. In *Proc. of the 3rd IJCAI*, pp. 235-245, Stanford, USA, 1973.

Priami, C. Computational thinking in biology, *Trans. on Comput. Syst. Biol. VIII*, LNBI 4780, pp. 63-76, Springer, 2007.

Schor, P. Algorithms for quantum computation: discrete logarithms and factoring. In *Proc. 35th Annual Symposium on Foundations of Computer Science*, IEEE Press, Los Almitos, CA, 1994.

Turing, A. On computable numbers with an application to the Entscheidungsproblem. In *Proc. London Mathematical Society* 42, pp. 230-265, 1936.

Wing, J. Computational thinking. *Comm. ACM*, pp. 33-35, March 2006.

DOI: 10.1145/1895419.1922682