DOI: 10.1145/2380656.2380658

Vinton G. Cerf

Computer Science Revisited

In a recent column, I questioned whether there was any "science" in computer science. This provoked a great many responses that provided some very valuable perspective. To the degree that some aspects of computing are subject to analysis and modeling, it is fair to say there is a rigorous element of science in our field. Computability, analytical estimates of work to factor numbers, estimates of parsing complexity in languages, estimates of the precision of computations, computational costs of *NP*-complete problems, among others, fall into the category I call "analytic" in the sense we can say fairly strongly how computational cost grows with scale, for example. In recent conversations with Alan Kay, Edward Feigenbaum, Leonard Kleinrock, and Judea Pearl, I have come to believe that certain properties contribute to our ability to analyze and predict behaviors in software space.

Alan Kay spoke eloquently at a recent event honoring Judea Pearl's ACM Turing Award. Among the points that Alan made was the observation that computing is not static. Rather it is a dynamic process and often is best characterized as interactions among processes, especially in a networked environment. The challenge in understanding computational processes is managing the explosive state space arising from the interaction of processes with inputs, outputs, and with each other.

In a recent discussion with Edward Feigenbaum, he noted that models used in hardware analysis are more tractable than those used in software analysis because what is possible in the "physical" world necessarily constrains the search for solutions. Software systems are designed and analyzed in "logical" space, in which there is far less structure to exploit in the analysis. However, the enormous computing speeds possible today, combined with some structuring of the software design space, might allow deep and effective analyses of software as yet unavailable to mathematical methods or human thought. He recalled how very fast search, structured by some knowledge of chess, allowed IBM's Deep Blue to defeat the world's chess champion Kasparov. In one game, Deep Blue, exploring hundreds of millions of possibilities in the analysis of a move, found a brilliant solution path, probably never before seen, that led Kasparov to resign from the game and lose of the match.

Structure that constrains state spaces is sometimes conferred through abstraction. Here we get to another key observation about the potential for scientific approaches to computer science. To the degree that abstraction eliminates unimportant details and reveals structure, abstraction is a powerful analytical tool. It strikes me that Chaos theory illustrates this notion because patterns emerge in this theory despite the apparent randomness of the processes it seeks to analyze. If programs and algorithms are subject to suitable abstraction, it may be possible to model them with adequate but abstract fidelity so as to render the model analyzable. An example is found in queueing theory in which complex processes are modeled as networks of queues. The models are analytic under the right conditions. Kleinrock made enormous contributions to network design and analysis through the construction of queueing theoretic models. His most famous contribution was the recognition that the state space explosion could be tamed by his Independence assumption in which the statistics of the traffic flowing through the network were regenerated at each node using statistically identical but independent variables.

Modeling is a form of abstraction and is a powerful tool in the lexicon of computer science. This leads me to a final observation illustrated by Judea Pearl's brilliant lecture on the use of Bayesian analysis to draw conclusions from problems involving probabilities rather than fixed values. Pearl's theories of causal reasoning in conditional probabilities are often aided by graph-like models linking the various conditional statements in chains of cause and effect. The diagrams make it possible to construct analytic equations that characterize the problem and make the solution computable. It struck me that Pearl's use of diagrams had an analogue in Richard Feynman's diagrams of quantum interactions. These diagrams are abstractions of complex processes that aid our ability to analyze and make predictions about their behavior. Pearl's diagrams may prove as powerful for science (not only computer science) as Feynman's have for quantum physics and cosmology.

I have come away from this foray into computer 'science' with several conclusions. The first is there really is science to be found in computer science. The second is abstraction and modeling are key to making things analytic. The third is there is a lot of complex detail in computer programs and collections of interacting programs that has not admitted much in the way of abstraction or modeling, rendering these complex systems difficult to analyze. Finally, I believe our ability to understand and predict software behavior may rest in the invention of better high-level programming languages that allow details to be suppressed and models to emerge. I hope Alan Kay's speculation will lead to serious improvements in the way we design and program computer systems.

Vinton G. Cerf, ACM PRESIDENT