DOI:10.1145/1965724.1965725

Moshe Y. Vardi

Solving the Unsolvable

On June 16, 1902, British philosopher Bertrand Russell sent a letter to Gottlob Frege, a German logician, in which he argued, by using what became known as "Russell's Paradox," that Frege's logical system was inconsistent. The letter launched a "Foundational Crisis" in mathematics, triggering an almost anguished search for proper foundations for mathematics. In 1921, David Hilbert, the preeminent German mathematician, launched a research program aimed at disposing "the foundational questions once and for all." Hilbert's Program failed; in 1931, Austrian logician Kurt Goedel proved two incompleteness theorems that proved the futility of Hilbert's Program.

One element in Hilbert's Program was the mechanization of mathematics: "Once a logical formalism is established one can expect that a systematic, so-to-say computational, treatment of logic formulas is possible, which would somewhat correspond to the theory of equations in algebra." In 1928, Hilbert and Ackermann posed the "Entscheidungsproblem" (Decision Problem), which asked if there is an algorithm for checking whether a given formula in (first-order) logic is valid; that is, necessarily true. In 1936–1937, Alonzo Church, an American logician, and Alan Turing, a British logician, proved independently that the Decision Problem for first-order logic is *unsolvable*; there is no algorithm that checks the validity of logical formulas. The Church-Turing Theorem can be viewed as the birth of theoretical computer science. To prove the theorem, Church and Turing introduced computational models, recursive functions, and Turing machines, respectively, and proved that the *Halting Problem*—checking whether a given recursive function or Turing machine yields an output on a given input—is unsolvable.

The unsolvability of the Halting Problem, proved just as Konrad Zuse in Germany and John Atanasoff and Clifford Berry in the U.S. were embarking on the construction of their digital computers—the Z3 and the Atanasoff-Berry Computer—meant that computer science was born with a knowledge of the inherent limitation of mechanical computation. While Hilbert believed that "every mathematical problem is necessarily capable of strict resolution," we know that *the unsolvable* is a barrier that cannot be breached. When I encountered unsolvability as a fresh graduate student, it seemed to me an insurmountable wall. Much of my research over the years was dedicated to delineating the boundary between the solvable and the unsolvable.

It is quite remarkable, therefore, that the May 2011 issue of *Communications* included an article by Byron Cook, Andreas Podelski, and Andrey Rybalchenko, titled "Proving Program Termination" (p. 88), in which they argued that "in contrast to popular belief, proving termination is not always impossible." Surely they got it wrong! The Halting Problem (termination is the same as halting) is unsolvable! Of course, Cook et al. do not really claim to have solved the Halting Problem. What they describe in the article is a new method for proving termination of programs. The method itself is not guaranteed to terminate—if it did, this would contradict the Church-Turing Theorem. What Cook et al. illustrate is that the method is remarkably effective in practice and can handle a large number of real-life programs. In fact, a software tool called Terminator, used to implement their method, has been able to find some very subtle termination errors in Microsoft software.

I believe this noteworthy progress in proving program termination ought to force us to reconsider the meaning of unsolvability. In my November 2010 editorial, "On P, NP, and Computational Complexity," I pointed out that NP-complete problems, such as Boolean Satisfiability, do not seem as intractable today as they seemed in the early 1970s, with industrial SAT solvers performing impressively in practice. "Proving Program Termination" shows that unsolvable problems may not be as unsolvable as we once thought. In theory, unsolvabilty does impose a rigid barrier on computability, but it is less clear how significant this barrier is in practice. Unlike Collatz's Problem, described in the article by Cook et al., most real-life programs, if they terminate, do so for rather simple reasons, because programmers almost never conceive of very deep and sophisticated reasons for termination. Therefore, it should not be shocking that a tool such as Terminator can prove termination for such programs.

Ultimately, software development is an engineering activity, not a mathematical activity. Engineering design and analysis techniques do not provide mathematical guarantee, they provide confidence. We do not need to solve the Halting Problem, we just need to be able to reason successfully about termination of real-life programs. It is time to give up our "unsolvability phobia." It is time to solve the unsolvable.

Moshe Y. Vardi, EDITOR-IN-CHIEF