DOI:10.1145/2093548.2093549

Moshe Y. Vardi

What is an Algorithm?

The 14th International Congress of Logic, Methodology and Philosophy of Science (CLMPS), held last July in France, included a special symposium on the subject of "What is an algorithm?"

This may seem to be a strange question to ask just before the Turing Centenary Year, which is now being celebrated by numerous events around the world (see http://www.turingcentenary.eu/). Didn't Turing answer this question decisively? Isn't the answer to the question "an algorithm is a Turing machine"?

But conflating algorithms with Turing machines is a misreading of Turing's 1936 paper "On Computable Numbers, with an Application to the Entscheidungsproblem." Turing's aim was to define *computability*, not algorithms. His paper argued that every function on natural numbers that can be computed by a human computer (until the mid-1940s a computer was a person who computes) can also be computed by a Turing machine. There is no claim in the paper that Turing machines offer a general model for algorithms. (See S. B. Cooper's article on page 74 for a perspective on Turing's model of computation.) So the question posed by the special CLMPS symposium is an excellent one.

Incontrovertibly, algorithms constitute one of the central subjects of study in computer science. Should we not have by now a clear understanding of what an algorithm is? It is worth pointing out that mathematicians formalized the notion of proof only in the 20th century, even though they have been using proofs at that point for about 2,500 years. Analogously, the fact that we have an intuitive notion of what an algorithm is does not mean that we have a formal notion. For example, when should we say that two programs describe the *same* algorithm? This requires a rigorous definition!

Surprisingly, the two key speakers in the symposium, Y. Gurevich and Y. Moschovakis, provided very different answers to this basic question, even after focusing on classical sequential algorithms (see in-depth papers at http://goo.gl/0E7wa and http://goo.gl/HsQHq).

Gurevich argued that every algorithm can be defined in terms of an *abstract state machine*. Intuitively, an abstract state machine is a machine operating on states, which are arbitrary data structures. The key requirement is that one step of the machine can cause only a bounded local change on the state. This requirement corresponds both to the one-cell-at-a-time operations of a Turing machine and the bounded-width operations of von Neumann machines.

Moschovakis, in contrast, argued that an algorithm is defined in terms of a *recursor*, which is a recursive description built on top of arbitrary operations taken as primitives. For example, the factorial function can be described recursively, using multiplication as a primitive algorithmic operation, while Euclid's greatest-common divisor algorithm can be described recursively, using the remainder function as a primitive operation.

Is an algorithm an abstract state machine or a recursor?

So is an algorithm an abstract state machine or a recursor? Mathematically, one can show that recursors can model abstract state machines and abstract state machines can model recursors, but which definition is primary?

I find this debate reminiscent of the endless argument in computer science about the advantages and disadvantages of imperative and functional programming languages, going all the way back to Fortran and Lisp. Since this debate has never been settled and is unlikely to be ever settled, we learned to live with the coexistence of imperative and functional programming languages.

Physicists learned to live with de Broglie's *wave-particle duality*, which asserts that all particles exhibit both wave and particle properties. Furthermore, in quantum mechanics, classical concepts such as "particle" and "wave" fail to describe fully the behavior of quantum-scale objects. Perhaps it is time for us to adopt a similarly nuanced view of what an algorithm is.

An algorithm is *both* an abstract state machine and a recursor, and neither view by itself fully describes what an algorithm is. This *algorithmic duality* seems to be a fundamental principle of computer science.

Moshe Y. Vardi, EDITOR-IN-CHIEF