Programmability

Wendy Hui Kyong Chun

According to the *Oxford English Dictionary*, programmability is "the property of being programmable"—that is, capable of being programmed.[1] Although the term "program," as both noun and verb, predates the modern digital computer, programmability and programmable do not, and the digital computer has changed the meanings of the word "program." The definition of program, the noun, not only includes "a descriptive notice, issued beforehand of any formal series of proceedings" and a "broadcast presentation treated as a single item for scheduling purposes" but also "a series of coded instructions, which when fed into a computer will automatically direct its operation in carrying out a specific task."[2] Program, the verb, includes "to arrange by or according to a programme," and "to broadcast," as well "to express (a task or operation) in terms appropriate to its performance by a computer or other automatic device; to cause (an activity or property) to be automatically regulated in a prescribed way."[3] Combined with the fact that "stored program" has become synonymous with von Neumann architecture, these definitions make it appear that programs are native to computers. Programs, however, were not always programs.

As David Alan Grier, among others, has argued, the term program did not stabilize until the mid-1950s.[4] According to Grier, the verb "to program" is probably the only surviving legacy of the ENIAC—the first working electronic digital computer and the immediate precursor to those using stored programs. Importantly, the ENIAC's "master programmer" was not a person, but a machine component, responsible for executing loops and linking sequences together.[5] That is, the master programmer handled the "program control" signal that each unit produced after it successfully executed a function. This use of "program" stems from electronics engineering, where a program signal is any signal corresponding to speech, or other activity, and stresses program as a thing that is transmitted, rather than a thing responsible for execution. As computers became machines, programmers became human and programming became functionally equivalent to the process of "setting up" the ENIAC—the physical act of wiring the machine for a particular problem.[6] Indeed, this "setting up" (once considered "operating" the machine) has been retroactively classified as "direct programming"; Grier argues that the term "program" was favored over the term "planning" (then in use in numerical methods) in order to distinguish machine from human computing. Although this process of setting up the machine seems analogous to the same process on contemporaneous analog machines, there is an important difference between them, for programmability marks the difference between digital and analog machines. This is not to say that analog machines are not programmed, but that what is meant by programming is significantly different. Programming an analog computer is descriptive; programming a digital one is prescriptive.

To program an analog machine, one connects the units and sets the values for amplification and attenuation, as well as the initial conditions. That is, one assembles the computer into an analog of the problem to be solved or simulated. As Derek Robinson argues: "while a digital computer can simulate feedback processes by stepwise iteration . . . analog computers embody dynamic feedback fundamentally. The "computation" takes place at all points in the circuit at the same time, in a continuous process. Circuits are systems of circular dependencies where effects are fed back to become the causes of their own causes."[7] That is, analog computers perform integration directly and can be used "generatively." Digital computers, on the other hand, employ numerical methods. They break down mathematical operations, such as integration, into a series of simple arithmetical steps. To do so, they must be programmable; that is, they must be able to follow precisely and automatically a series of coded instructions. Although one would think that the breakdown of mathematical operations into a series of arithmetical ones would induce more errors than direct integration, this is not usually so. The programmability and accuracy of digital computers stems from the discretization (or disciplining) of hardware.

Since analog computers produce signals that simulate the desired ones, they are measured rather than counted. The accuracy of the result is thus affected by noise and its precision by the sensitivity of the measuring instrument. In contrast, digital computers count rather than measure, and they do so by rendering analog hardware (vacuum tubes, transistors, etc.) signal magnitudes into discrete units. By translating a certain quantity into a value (5V into 1; 0V into 0), they can greatly reduce the effects of noise, and thus essentially build a system in which one step can predictably lead to another. As Alan Turing and von Neumann both acknowledged early on, there are no "discrete" or digital machines; there are only continuous machines that in Turing's words can be "profitably thought of as being discrete state machines," machines in which, "given the initial state of the machine and the input signals it is always possible to predict all future states." This, he argues, "is reminiscent of Laplace's view that from the complete state of the universe at one moment of time, as described by the positions and velocities of all particles, it should be possible to predict all future states."[8] Again, reasonably accurate results depend on the design of hardware in specific ways: on timing gates carefully so that gate delays do not produce significant false positives or negatives; on signal rectification; and on designs that cut down on cross-talk and voltage spikes. Without this disciplining of hardware, digital computers—or digital-analog hybrids—could not be (however inadequately or approximately) universal mimics, or Turing complete.

This "return to Laplace" and the desire for programmability (and programs as we now know them) was arguably predated by work in mid-twentieth-century genetics. Most famously, Erwin Schrodinger, drawing from the work of contemporaneous researchers in biology and chemistry, posited the existence of a genetic code-script in his 1944 *What is Life?*[9] Schrodinger posits this code-script as the answer to the challenge human genetics presents to statistical physics, namely, given that statistical physics shows that Newtonian order only exists at large scales, how is it possible that the barely microscopic chromosomes guarantee the orderly succession of human characteristics? Also, given the second law of thermodynamics, how does life maintain order in this sea of disorder? Given how microscopic the chromosomes are, Schrodinger argues that they must be an aperiodic crystal code-script, a code-script—not unlike Morse code—that determines the entire pattern of an individual's future development and of its functioning in the mature state. Thus the code-script is a seemingly impossible return to Laplace. Schrodinger writes, "in calling the structure of the chromosome fibres a code-script we mean that the all-penetrating mind, once conceived by Laplace, to which every causal connection lay immediately open, could tell from their structure whether the egg would develop, under suitable conditions, into a black cock or into a speckled hen, into a fly or a maize plant, a rhododendron, a beetle, a mouse or a woman."[10] Importantly, software—which was not foreseen by computing pioneers—and not DNA would come to fulfill Schrodinger's vision of a code-script as "architect's plan and builder's craft in one."

Just as Schrodinger links programmability to an all-penetrating mind, programmability is linked to the feelings of mastery attributed to programming, its causal pleasure.[11] As Edwards has argued, "programming can produce strong sensations of power and control" because the computer produces an internally-consistent if externally incomplete microworld, "a simulated world, entirely within the machine itself, that does not depend on instrumental effectiveness. That is, where most tools produce effects on a wider world of which they are only a part, the computer contains its own worlds in miniature. . . In the microworld, as in children's make-believe, the power of the programmer is absolute."[12] This power of the programmer, however, is not absolute and there is an important difference between the power of the programmer/programming and the execution of the program. Alan Turing, in response to the objection that computers cannot think because they merely follow human instructions, wrote, "machines take me by surprise with great frequency."[13] This is because the consequences of one's programs cannot be entirely understood in advance. Also, as Matthew Fuller has argued in his reading of Microsoft Word, there is an important gap between the program and the experience of using it. The mad attempt to prescribe and anticipate every desire of the user produces a massive feature mountain whose potential interaction sequences mean that a user's actions cannot be completely determined in advance: the more features a program provides, the more possibilities for the user to act unpredictably.[14]

Importantly, programmability is being attacked on all sides: from quantum computers that are set up rather than programmed (in the sense currently used in software engineering) to "evolutionary" software programs that use programmable discrete hardware to produce software generatively.[15] This apparent decline in programmability is paralleled in new understandings of genomics that underscore the importance of RNA (the same portion of DNA can transcribe more than one protein)—biology and computer technology are constructed metaphorically as two strands of a constantly unravelling double helix. This seeming decline, however, should not be taken as the death knell of programmability or control, but rather the emergence of new forms of control that encourage, even thrive on, limited uncertainty.[16]

Notes

1. Oxford English Dictionary Online (2006). http://www.oed.com/.

2. Ibid.

3. Ibid.

4. David Grier, "The ENIAC, the Verb 'to program' and the Emergence of Digital Computers," 51.

5. H. H. Goldstine and A. Goldstine, "The Electronic Numerical Integrator and Computer (ENIAC)," 10–15.

6. For more on this see Wendy Hui Kyong Chun, "On Software, or the Persistence of Visual Knowledge."

7. Derek Robinson, presentation at Software Studies Symposium, Piet Zwart Institute, Rotterdam, 2006.

8. Alan M. Turing, "Computing Machinery and Intelligence."

9. Erwin Schrodinger, *What is Life?*

10. Schrodinger, Ibid.

11. Chun, Ibid.

12. Paul Edwards, "The Army and the Microworld: Computers and the Politics of Gender Identity," 108–109.

13. Turing, Ibid.

14. Matthew Fuller, "It Looks Like You're Writing a Letter," in Behind the Blip, essays on the culture of Software.

15. For more on this see, N. Katherine Hayles, *My Mother was a Computer*; Julian Dibbell, "Viruses Are Good For You"; Jussi Parikka, "The Universal Viral Machine."

16. See Wendy Hui Kyong Chun, *Control and Freedom*.