Cognition, 23 (1986) 89–94 **Discussion**

When is a computer not a computer?

PETER KUGEL*
Boston College

We recently bought an IBM-BLOKBUSTR computer, after reading Osherson's (1985) beguiling description of its ability to do the uncomputable. We had hoped to use it in our study of the human ability to generate music. But we ran into problems that we had not anticipated. Readers of this journal who plan to use this rather unusual machine in their research, might want to profit from our experience.

What makes BLOKBUSTR different from other computers is that it can do more than compute. For example, it can run a program, K-BAR, that outputs all and only the integers in the set that mathematicians call "K-bar". K-bar is uncomputable. It consists of the indexes of Turing machines that do not halt when run on their own indexes. Since we can prove—mathematically—that this set cannot be generated by a computation, BLOKBUSTR can do something no ordinary computer can.

In our lab, we feel that some human cognitive abilities require more than computing. The ability to generate and recognize good music, for example. We planned to use BLOKBUSTR's ability to generate K-bar as a subroutine (or oracle) in a computer program that would simulate this human ability and generate music in the style of Mozart. Our aim was not to replace Mozart, who most people feel to be irreplaceable, but rather to try to characterize his style precisely. Like Stiny and Gips (1978), we feel that writing a program that generates works of art in a given style is one way to characterize that style precisely. Our plan to use K-bar was inspired by Myhill's (1952) suggestion that enumerating a set of beautiful things might require the ability to generate a productive set and the fact that K-bar is such a set. Our choice of Mozart, rather than the more popular (among those who generate computer music) Bach, was inspired by the fact that musicians call their enumeration of Mozart's work "K". At least they refer to his works as "K459", "K525" and the like. Since K-BAR enumerates the items not in the set that the mathematicians call "K", we thought it might be apt to use it to try to enumerate the set of those of Mozart's compositions not in the set the musicians

*Reprint requests should be sent to Peter Kugel, Computer Science Department, Boston College, Chestnut Hill, MA 02167, U.S.A.

Osherson (Osherson, Stob, & Weinstein, 1986), I am a fan of uncomputable models of the mind (Kugel, 1986). But I have serious reservations about our ability to build machines that do the uncomputable in a useful way. Fortunately, I don't think that it is necessary to build an "uncomputing machine" to use such machines as models for either computers or the mind.

There are at least three ways that a computer might do more than compute. One way is *incidentally*. A computer does something incidentally when it does so in ways that depend on accidental aspects, or "side effects", of its implementation. For example, a computer's printer might accidentally do something randomly that could not be fully duplicated by a computation. But, until we figure out how to harness such incidental behaviors to our purposes, they do not do anything we can use. True, they might require uncomputable models for their characterization and the rhythms of human speech might, as Osherson suggests, be such an incidentally uncomputable aspect of human behavior. Incidental uncomputable behavior is uncomputable but we may not be able to harness it to our purposes.

The second way that a computer can do more than compute is *usefully*. A computer does something usefully if we can use what it does to do things we want done. Computers might be able to do things incidentally without our being able to use what they do and what seems to be "broken" in our BLOKBUSTR is this link between what it does and our ability to use what it does. Perhaps its hardware is doing something uncomputable, but we cannot figure out how to use that behavior to get it to smudge the output as we want it smudged. Alas, Osherson seems to have forgotten to tell us how that link works.

The third way a computer might do more than compute, or "not be a computer" as the title of the note suggests, is *theoretically*. We do know how to get a machine to do the uncomputable theoretically *and* we know how to use it to do things that are (theoretically) useful. Theoretical trial and error machines, for example, produce uncomputable results that can do induction (Angluin & Smith, 1985). But trial and error machines are theoretical, and like Turing machines, they cannot be built. They require not only the unlimited memory of the Turing machines, but also unlimited time—a luxury that real people and real machines do not have available to them.

Neither Turing machines nor trial and error machines can be built, but real computers can approximate the behavior of both. There is, however, an important difference between the resulting approximations. When a computing machine approximates the outputs of a Turing machine, it does only some of the outputs correctly. Thus, for example, a real computer cannot even add the way a theoretical Turing machine adds. We expect our (theoretical) adding machines to be able to add any two numbers, no matter how large.

Given any real machine—and even super-computers are not exempt here—we can define a number so large that the machine, with all the memory resources at its command, cannot even represent one, let alone add two of them.

A computer's approximation of the results of a trial and error machine, however, are different. A trial and error machine differs from a computing machine in that we count its last output rather than its first and if the machine does not do us the courtesy of halting after its first output, we cannot always be sure when we have even one result in hand. For example, a trial and error machine, trying to prove Fermat's Last Theorem, by looking through all the positive integers for counterexamples, may tell us it "thinks" the theorem is true because it has not yet found a counterexample. But I would not be willing to rely on that result in anything like the way I am willing to trust the computer's conclusion that $1234 + 567 = 1801$. If there is some way we can get a trial and error result that is final in finite time (and if we could, we could enumerate K-bar), I wish somebody would show it to me. Nobody has, yet. And that is why I am doubtful about our ability to fix BLOKBUSTR.

Fortunately, we don't need to be able to build machines that do more than compute to use them as models. Many sciences work with models that cannot be realized in the actual world. For example, theoretical physics use bodies whose masses are concentrated at a single point, even though theoretical physicists know that such bodies are physically impossible. Science depends on theoretical models that can be studied mathematically, but it does not depend on our ability to build physical versions of these abstract models. Theoretical models are ways of looking at things, and not ways of building things.

They can be used to *guide* the way we build things. For example, Turing's study of the theoretical Turing machines led to the several ideas (the idea of a universal machine, the idea of the program) that in turn led to the building of real computers during the Second World War. Similarly, results about machines that can do more than compute might lead to ideas for using computers to do practical things in the real world. The results of Shapiro (1981), for example, might do precisely that.

Theoretically, computing machinery can be used to do more than compute. Perhaps physical computers can be made to do more than compute and do it in a way that we can use too. But I can't figure out how to go about it. Analog schemes, like those we tried to use in our ink, tend to violate laws of physics and to have "readout" problems. They often require actual infinitesimals, which cannot be physically realized, and it is hard to interpret their outputs with the kind of precision that doing the uncomputable seems to require. Trial and error schemes seem to give us results that we cannot

rely on after only finite time. Probabilistic schemes seem to give us either results that we cannot rely on or results that we cannot figure out how to use.

Still, the fact that the people in my lab cannot think of any way to fix BLOKBUSTR does not mean that it cannot be fixed. If it can, I can think of plenty of things to do with it and generating music is only one of them. Perhaps Osherson, or somebody else, can tell us how BLOKBUSTR works so that we can fix it. I certainly hope so. The National Science Foundation has withdrawn our funding and we cannot afford to buy a new manual.

References

Angluin, D., & C.H. Smith (1985). Inductive inference: theory and methods, *ACM Computer Surveys, 15* 237–270.

Gold, E.M. (1965). Limiting recursion, *Journal of Symbolic Logic, 30,* 28–48.

Kugel, P. (1986). Thinking may be more than computing, *Cognition, 22,* 137–198.

Myhill, J. (1952). Some philosophical implications of mathematical logic: three classes of ideas, *Review of Metaphysics, 6,* 165–198.

Osherson, D.N. (1985). Computer output, *Cognition, 20,* 261–264.

Osherson, D.N., M. Stob, & S. Weinstein (1986). *Systems that learn: An introduction to learning theory for cognitive and computer scientists.* Cambridge, MA: MIT Press.

Putnam, H. (1965). Trial and error predicates and the solution of a problem of Mostowski, *Journal of Symbolic Logic, 20,* 49–57.

Shapiro, E. (1981). Inductive inference of theories from facts, *Technical Report 192.* Computer Science Department, Yale University.

Stiny, G. & J. Gips (1978). *Algorithmic aesthetics.* Berkeley, CA: University of California Press.