The Halting Problem is Undecidable In Verse

By Geoffrey K. Pullum

No general procedure for bug checks succeeds.
Now, I won't just assert that, I'll show where it leads:
I will prove that although you might work till you drop,
you cannot tell if computation will stop.

For imagine we have a procedure called P
that for specified input permits you to see
whether specified source code, with all of its faults,
defines a routine that eventually halts.

You feed in your program, with suitable data,
and P gets to work, and a little while later
(in finite compute time) correctly infers
whether infinite looping behavior occurs.

If there will be no looping, then P prints out Good.
That means work on this input will halt, as it should.
But if it detects an unstoppable loop,
then P reports "Bad!" which means you're in the soup.

Well, the truth is that P cannot possibly be,
because if you wrote it and gave it to me,
I could use it to set up a logical bind
that would shatter your reason and scramble your mind.

Here's the trick that I'll use and it's simple to do.
I'll define a procedure, which I will call Q,
that will use P's predictions of halting success
to stir up a terrible logical mess.

For a specified program, say A, one supplies,
the first step of this program called Q I devise
is to find out from P what's the right thing to say
of the looping behavior of A run on A.

If P's answer is "Bad!", Q will suddenly stop.

But otherwise, Q will go back to the top,
and start off again, looping endlessly back,
till the universe dies and turns frozen and black.

And this program called Q wouldnt stay on the shelf;
I would ask it to forecast its run on itself.
When it reads its own source code, just what will it do?
What's the looping behavior of Q run on Q?

If P warns of infinite loops, Q will quit;
yet P is supposed to speak truly of it!
And if Q's going to quit, then P should say "Good"
which makes Q start to loop! (P denied that it would.)

No matter how P might perform, Q will scoop it:
Q uses Ps output to make P look stupid.
Whatever P says, it cannot predict Q:
P is right when its wrong, and is false when its true!

I've created a paradox, neat as can be
and simply by using your putative P.
When you posited P you stepped into a snare;
Your assumption has led you right into my lair.

So where can this argument possibly go?
I don't have to tell you; I'm sure you must know.
By reduction, there cannot possibly be
a procedure that acts like the mythical P.

You can never find general mechanical means
for predicting the acts of computing machines.
Its something that cannot be done. So we users
must find our own bugs. Our computers are losers!

NOTE: In October 2000, after a refereeing delay of nearly a year, an earlier and incorrect version of this poetic proof was published in *Mathematics Magazine* (73, no. 4, 319-320). I am very grateful to Philip Wadler (Informatics, University of Edinburgh) and Larry Moss (Mathematics, Indiana

University) for helping me to develop this corrected version, which is now free of bugs (trust me; you can check it). Thanks also to the late Dr. Seuss for the style, and of course to the pioneering work of Alan Turing (and Martin Daviss nice simplified presentation) for the content.