

Perl And Undecidability

The Halting Problem

by Jeffrey Kegler

jeffreykegler@mac.com



How do I know what a Perl program is going to do without running it? Does it matter that it's Perl instead of some other language? It turns out that the answer is related to the Halting Problem, which says that there is no *general* solution to the question of whether an arbitrary computer program will ever stop running. If I can't decide that, I can't perfectly analyze source code to figure out what it is doing without running it.

I first started thinking about this problem because it's a thorn-in-the-side for Adam Kennedy's PPI module, which attempts to parse Perl code without running it. PPI is the backbone of `Perl::Critic`, a tool to enforce code policy. Can PPI ever be 100% correct? I won't answer that question in this article, but I will go through the Halting Problem to show you how it applies to Perl. This is a first in a series of three articles that will eventually give you that answer.

These articles will avoid mathematical notation in favor of Perl 5. Perl 5 has its disadvantages for this work but it is an excellent way of presenting algorithms, in many ways much less awkward than the mathematical notations. I hope some readers, seeing the ideas in this form, will become comfortable enough with them to tackle them in more standard notation. If you decide you want to do that, Wikipedia is an excellent place to start.

■ Undecidability-----

A computer problem is *undecidable* if it's a decision about which of two sets an input falls into, and it's impossible to make that decision. Every computer problem can be seen as one or more decisions between two choices, so that all limits to decidability are limits to what can be done by computer. This is the first in a series of three articles about an insight that undecidability gives us into Perl, and the insights Perl gives us into undecidability.

These days computability and undecidability are treated as synonyms, but questions about decidability predate our modern notion of computing. The first person to formalize computability was Alan Turing, who introduced the Halting Problem. The Halting Problem is the problem of deciding whether a given computer with a given input runs forever or eventually halts. Turing didn't call it the Halting Problem, but since his day it's gotten that name. Turing showed that the Halting Problem was uncomputable and therefore undecidable.

The Halting Problem is far from the only undecidable problem. There are many others, and it is not at all hard to run into them in everyday practice. It is usually obvious that an undecidable problem is going to be, at the least, very difficult. You often avoid them anyway, and in that case whether a problem

is undecidable or merely just plain hard is a quibble. But there are also problems whose solution, if possible, justifies major resources. Knowing a problem is undecidable allows you to either redefine it or give up.

■ Borrow your cat? -----

There's a big problem with writing a proof in Perl. When I see Perl code, it raises a set of expectations. Most of those expectations are wrong when it comes to proofs. Perl code is usually written to *do* something. This code with this article is not: it's written to explore an idea as a thought problem.

The most famous example of a thought problem is Schrödinger's cat. In it, you need a cat. You lock him in a box shielded not only from outside interference, but from outside observation. Inside this box you put another box. This inner box is not airtight, but it is shielded from interference by the cat. The inner box has a geiger counter triggered to detect atomic decays. The geiger counter has a relay which controls a hammer hanging over a flask, closed, fragile, and filled with a gas of high feline toxicity. Dr. Schrödinger suggested hydrocyanic acid.

According to quantum mechanics, the geiger counter, as long as it is not observed, never trips the relay, never releases the hammer, never breaks the glass, and never kills the cat. But neither is the opposite true. You cannot say that the relay was tripped, that the hammer dropped, that the flask was broken, or that the cat is dead. The situation inside the box is a mixture of the two, not even a probability, but a kind of probability wannabe called a quantum superposition. When the box is opened the superposition collapses into a probability, the probability collapses into a fact, and the cat either either collapses dead or springs out of the box.

So what's up with the cat before the box is opened? That's the point of the problem. It doesn't really engage our intuition to be told that the decay state of an atom is a mathematical function involving complex numbers. To be told that about the trace a geiger counter leaves on its recording tape is a bit more of a challenge. But neither compares to the challenge involved in trying to envision a cat who is not alive, not dead, not in a transition from life to death, not in a transition from death to life, and not even in a state which is a red-blooded real-numbered probability of life or death. Now that's a challenge to our intuition.

If you think about Schrödinger's cat like as an engineer, you will ask questions like: If the box is airtight, how does the cat breathe? If the box is not airtight, does the cat's respiration interact with the outside world? Doesn't that ruin the experiment? How do you calibrate the geiger counter? Since hydrocyanic