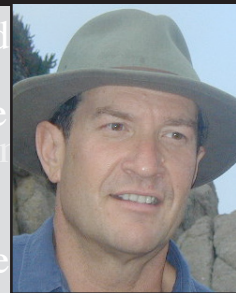


Perl and Undecidability: Rice's Theorem

by Jeffrey Kegler

jeffreykegler@mac.com



This winter I moved into a cabin at the edge of a frozen lake and forsook gainful employment in favor of work on a CPAN module. Currently, no generally accepted, handy tool accepts arbitrary BNF and parses with it. Recent research suggests how to create that tool. I call my effort `Parse::Marpa`. An alpha version is now on CPAN.

Approaching beta, I thought about potential applications. My thoughts turned to Perl 5. The Perl community had reached the consensus that Perl 5 is not statically parseable. Adam Kennedy, in the documentation for PPI, indicated how this might be proved. I worked the proof out formally and posted it on Perlmonks.

The formal proof shows that Perl 5 is not just statically unparseable, it is also dynamically unparseable. The saying had been that “Only perl can parse Perl”. In fact, not even Perl 5 can parse Perl 5 in every case.

Here's the reason: The only way to parse Perl 5 is to run it or to simulate it using a language of equivalent power. Perl 5 is what's called Turing-complete, and all Turing-complete languages are subject to the Halting Problem. There is no guarantee a Perl 5 program will ever finish running and no guarantee it will ever finish parsing itself.

In this article I will use Rice's Theorem to prove that Perl is unparseable. Rice's Theorem is powerful and easy to apply. It's well known in mathematical circles and deserves to be better known by programmers.

■ Undecidable parsing is a feature-----

Undecidable parsing is not a bug. It is not a misfeature. It goes hand-in-hand with important capabilities. Understanding this is important to understanding where programming languages are going.

Perl 5 is unparseable because it gives the programmer Turing-complete power before compile time. As time goes on, it becomes clearer and clearer that Larry Wall aimed Perl in the right direction. Theoretical perfection at compile time is a loss at run-time. Industrial strength debugging and optimization require information unavailable before run-time. Decidability is not good if the decisions are bad.

■ Decidability-----

Decidability means the ability for a Turing-complete machine (or language), to determine the answer to a yes/no question. A yes/no question is decidable if a Turing-complete Perl script can answer it. Otherwise it is undecidable.

Turing-equivalence means equivalence to the model of computing in Alan Turing's 1936 paper. A machine or language is Turing-complete if it has Turing-equivalent power or better. All modern general-purpose machines and languages are at least Turing-complete.

Perl scripts differ from theoretical Turing-complete programs in two ways, neither of them serious obstacles to Rice's Theorem. First, Turing-complete machines and programs have unlimited memory. The equivalent Perl implementation would never run out of memory. A real-life Perl script will fail to answer an undecidable question by running out of memory. Its theoretical Turing-complete counterpart fails by running forever. This is not a difference we need to care about.

Second, Perl scripts have certain capabilities which Turing-equivalent machines do not. Turing-equivalent machines and languages must be completely predictable (deterministic). Perl has unpredictable features like its `rand` built-in. Pedantically, `rand` is pseudo-random instead of random, but from the point of view of the Perl script `rand` is close enough to random. Perl's ability to interact with outside processes and across networks means many Perl calls behave unpredictably, perhaps even in the quantum mechanical sense.

But unpredictability is no obstacle to undecidability proofs. An undecidable question does not become decidable when its subject matter becomes unpredictable.

■ Undecidability-----

Rice's Theorem states that every interesting question about what a Perl script does is undecidable.

- Does a Perl script ever print the character '0'?
- Does a Perl script write to STDERR?
- Is a Perl script's output the same as its input?
- Does a Perl script fork a shell?
- Does a Perl script contain a virus?

All of these questions can be proved to be undecidable using Rice's Theorem. Here is Rice's Theorem more formally:

Any question about what an arbitrary Perl script does with an arbitrary input is undecidable, unless it is trivial.

■ Trivial? -----

For the purposes of Rice's Theorem, a question is trivial if the answer is always “yes”, or if the answer is always “no”. A