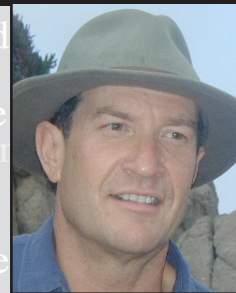


# Perl Is Undecidable

by Jeffrey Kegler

jeffreykegler@mac.com



This is the last of three articles based on a formal proof of Perl's unparseability that I originally presented on Perlmonks. This winter I retired to the edge of a frozen New England lake to work full-time on a parser generator, one that would generate a parser from any grammar describable in BNF. No such tool is in general use. I hope to create one. An alpha version is on CPAN as `Parse::Marpa`.

Looking for test cases, I considered Perl 5. This led me to Adam Kennedy's PPI documentation and his suggestion of how to prove that Perl parsing is not decidable. For some reason, it was not immediately obvious to me that Adam was right. Perhaps my optimism about parsing Perl 5 came from routinely walking on water (frozen lake, remember). In any case, to be convinced, I needed to work the proof out formally for myself.

I posted the result on Perlmonks. Since the Perl unparseability proof is of practical interest rather than theoretical significance, I tried to make my write-up accessible to readers who don't care about math for its own sake, but who do care about things that are of practical use. The Perlmonks posting attracted enough interest for me to be invited to write this series.

The first of these articles proved the Halting Theorem using Perl notation. One purpose was to explain the techniques I would need in the other two articles, but it should not be forgotten that the Halting Theorem itself is an extremely practical and useful result. Imagine for a moment that the existence of unsolvable problems was known only to experts in universities. Imagine in particular that it was not generally known that I can't write a program to find infinite loops in arbitrary code. A lot of time would be wasted.

The second article dealt with Rice's Theorem, a quick and handy rule for spotting undecidable problems. Most programmers know that there are undecidable problems, and that some of them are practical questions. Less well known is just how common undecidability is. Any non-trivial question about what a Perl script does is undecidable, and the same is true of all general-purpose programming languages.

## ■ The proof by way of Rice's Theorem -----

Perl is unusual among general-purpose languages in that not just Perl's run phase behavior, but also its parsing is, in the general case, undecidable. The second article contained a Perl unparseability proof. The proof used Rice's Theorem, which had several advantages.

1. It was short and quick.
2. It is closest to how the proof would look in a journal if it were a publishable result. (An academic math journal would not print this result because the referees would consider it obvious. They would also reject it because practical programming languages can have limited lifespans, and the journals want results that will be relevant and readable, decades from now.)
3. Using Rice's Theorem, the second article also proved that a wide variety of other questions about Perl were undecidable, showing that the situation with Perl parsing and the Halting Question is far from rare.

The disadvantage of using Rice's Theorem is that it is a bit of a "black box". It might leave me without any feeling for why Perl is not in general parseable.

This article presents two more proofs. Each lifts the cover of the black box. The first is direct, that is, it avoids the traditional approach of reduction to the Halting Theorem. Instead it assumes the existence of a general solution to Perl parsing and uses an example Perl script to show that this assumption simply can't be true. The second proof in this article takes the traditional approach, showing that a general solution to Perl parsing requires a general solution to the Halting Question, which is known (and which was proved in the first article of this series) to be undecidable.

Both the proofs in this article employ a reduction to absurdity—they assume something, and show that the assumption creates a contradiction. This constitutes a proof that the assumption must be false, and that therefore the opposite of the assumption must be true. Ordinary reasoning uses this kind of logic all the time ("If this jerk knows so much about startups, how come he needs us to pick up the tab for lunch?"). But for some reason, when presented in its raw form, reduction to absurdity can seem strange.

*See parts I and II of this series  
in The Perl Review,  
Spring 2008 and Summer 2008*