The Halting Problem in Forth

Bill Stoddart

September 1, 2016

Abstract

Forth can be used to formulate a simplified but fully general statement of the halting problem and to formulate a short and simple proof.

Keywords: Forth, halting problem, proof

1 Introduction

In his 1936 paper "On Computable Numbers", Alan Turing formulated the idea of a Turing machine and its tape as a way of describing "effective procedure" and showed that there were some limitations in such machines. A slight variation in the limitations that Turing demonstrated gave us the "Halting Problem". This phrase may hafirst have appeared in the 1958 textbook *Computability and Unsolvability* by Martin Davis, but the problem is generally attributed to Turing due to its closeness to the material in his paper. It shows that no Turing machine can exist such that, if supplied with the description of another arbitrary Turing machine S and its data (tape) D, would be able to predict whether S would eventually come to a halt if activated on D. Given that we recognise Turing machines as representing computations in a general sense, it tells us that no program can be written which can take another program S and data D as input and reliably tell us whether S will halt when executed on data D.

In this paper we first formulate a slight simplification of the halting problem in Forth. We then discuss what the mathematical implications would be if a solution to the halting problem, in the form of some program H, *did exist*, showing that this would provide a effective procedure for demonstrating the truth of mathematical propositions.

We then give a Forth based proof of the halting problem, and set a variation of the problem as an exercise.

2 Describing the halting problem in Forth

When a Forth program is executed from the keyboard it either comes back with an “ok” response, or exhibits some pathological behaviour such as reporting an error, not responding because it is in a n infinite loop, or crashing the whole system. We classify the “ok” response as what we mean by “halting”.

We specify a putative Forth program H by its stack effect:

$xt \rightarrow f$, f will be true if and only if execution of xt from the current state would halt.

Note that we do not talk about the application of a program to its data, but it is implicit that there is a stack where any data required by the execution of `xt` may be found.

Were H to exist, we could use it as follows:

$4\ 2\ '/\ H\ .\ \hookleftarrow -1\ ok$
$4\ 0\ '/\ H\ .\ \hookleftarrow 0\ ok$

3 Implications of the existence of H

Fermat’s last theorem states that for any integer $n > 2$ there are no integers a, b, c such that:

$$a^n + b^n = c^n$$

Fermat died leaving a note in the margin of his notebook saying he had found a truly marvellous proof of his theorem, but this proof was never found. All subsequent attempts proof failed until 1995, when Andrew Wiles produced a proof 150 pages long.

However, with the aid of our program `H` we could have investigated Fermat’s last theorem by providing a Forth program *FERMAT* which searches exhaustively for a counter example and halts when it finds one. Then we could have proved the theorem by the execution:

′ FERMAT H . ↲ 0 *ok*

This tells us the program *FERMAT* does not halt, implying that the search for a counter example will continue forever, in other words that no counter example exists and the theorem is therefore true.

In the same way we could explore any mathematical conjecture by writing a program to search exhaustively over the variables of the conjecture until a counter example is found. Then use H to determine if the program fails to halt, in which case there is no counter example, and the conjecture is proved.

4 A proof of the halting problem in Forth

Traditional proofs of the halting problem and friends rely on a diagonalisation argument - see §8 of Turing's paper. We will permit ourselves a more direct approach.

We assume a program H exists with stack effect $xt \rightarrow f$ where f will be true if execution of xt halts, and false otherwise.

We specify a program *IH* (*xt* →) which inverts the halting behaviour of *xt*, i.e. it halts if execution of *xt* would fail to halt, and it fails to halt if execution of *xt* would halt. We can define this program by:

: *IH* (*xt* →) *DUP H IF BEGIN AGAIN THEN* ;

We then consider whether *′ IH IH* will halt.

When we execute *′ IH IH* the invocation of H within *IH* finds *′ IH ′ IH* on the stack, so it will report whether *′ IH IH* will halt.

If we assume H returns true, reporting that *′ IH IH* will halt, then *IH* will enter a non-terminating loop, so we must discard this assumption.

If we assume H returns false, reporting that *′ IH IH* will not halt, the invocation of H in *IH* must have yielded false, which would yield immediate termination. Again we must discard this assumption.

So we are forced to reject our assumption that the program H exists.

5 A similar non-existence proof for the zero test program - exercise

Turing considered a the analysis of a slightly different machine, one supposed to tell whether a Turing machine with a given tape will ever output a specific symbol, say a zero, to that tape.

We adapt this to Forth by investigating whether a program Z could exist with this specification:

$xt \ \rightarrow \ f$, f will be true if an only if execution of xt leaves a zero at the top of the stack.

Exercise: Prove that no such program Z can exist.

6 Conclusion

To demonstrate the halting problem in Forth we assume the existence of a program $H \ \ xt \ \rightarrow f$, f is true if and only if execution of xt halts. We then use H to define a program:

$: \ IH \ (xt \ \rightarrow) \ DUP \ \ H \ \ IF \ \ BEGIN \ AGAIN \ \ ;$

and we show IH cannot exist by a reductio ad absurdum obtained from considering execution of $' \ IH \ \ IH$.

This approach is very uncluttered, due to the minimalism of Forth, but also differs from other approaches to the halting problem in that it does not require formulation in terms of a program acting on given data - with our approach, using a stack, the presence of any data required for our arguments can be left implicit.