Halting Problem

Eric C.R. Hehner

Department of Computer Science, University of Toronto
hehner@cs.utoronto.ca

Abstract Halting cannot be computed due to inconsistency of specification.

Problem

The first proof that halting is incomputable was by Alan Turing in 1936 [1]. I begin by presenting Turing's proof, but modernized in two respects. Instead of using Turing Machine operations, I use a modern programming language: Pascal. To apply a function to the domain of programs, Turing encoded programs as numbers. Today, when one program is presented as input data to another program (for example, to a compiler or interpreter), it is represented as a text (character string), and that's the encoding I will use. These changes are standard in modern textbooks; they change nothing essential in the proof of incomputability.

I cannot write a Pascal function to say whether the execution of any Pascal procedure halts, so instead I write the function header, and a comment to specify what the result of the function is supposed to be. Following the program is the standard argument leading to the incomputability conclusion.

```
function halts (p, i: string): boolean;
{ returns true if string p represents a Pascal procedure with one string input }
{ whose execution halts when given input i ; returns false otherwise }

procedure twist (s: string);
begin
        if halts (s, s) then twist (s)
end
```

Assume function *halts* has been programmed according to its specification/definition/description/comment. Does execution of *twist* ('*twist*') terminate? If it terminates, then *halts* ('*twist*', '*twist*') returns *true* according to its specification, and so we see from the body of *twist* that execution of *twist* ('*twist*') does not terminate. If it does not terminate, then *halts* ('*twist*', '*twist*') returns *false* , and so execution of *twist* ('*twist*') terminates. This is a contradiction (inconsistency). Therefore function *halts* cannot have been programmed according to its specification; *halts* is incomputable.

I agree with the first half of the concluding sentence: function *halts* cannot be programmed according to its specification. But I claim that the reason is not due to incomputability; it is due to an inconsistency (or self-contradiction) in the specification.

It is easy to write procedures for which the desired result of *halts* is clear. For example:

```
procedure fine (s: string);
begin
    if s='loop' then fine (s)
end
```

The result of *halts* ('*fine*', '*fine*') should be *true* , and the result of *halts* ('*fine*', 'loop') should be *false* . Examples like this may give us false comfort that *halts* is consistently defined. But *halts* must apply to all Pascal procedures with one string input, including *twist* . So what should the result of *halts* ('*twist*', '*twist*') be? I am not asking how to compute it; I just want to know what the result should be. If we say it should be *true* , then execution of *twist* ('*twist*') is nonterminating, so *halts* ('*twist*', '*twist*') should be *false* . If we say it should be *false* , then execution of *twist* ('*twist*') is terminating, so *halts* ('*twist*', '*twist*') should be *true* . Both options are ruled out. This is an inconsistency in the specification, or definition, of *halts* .

Simpler Version

Now I want to get rid of a distraction in the previous presentation of the Halting Problem. Although *halts* is defined to say whether execution of any Pascal procedure *p* terminates when given any input *i* , the argument for incomputability uses *halts* for only one procedure, *twist* , on only one input, '*twist*' . So now let me write a simpler version of *halts* that works on only the one procedure we're interested in, eliminating the parameters. This simplified version supports the argument leading to incomputability just as well (or badly) as the previous version.

```
function halts: boolean;
{ returns true if execution of twist halts; returns false otherwise }

procedure twist;
begin
    if halts then twist
end
```

Assume function *halts* has been programmed according to its specification. Does execution of *twist* terminate? If it terminates, then *halts* returns *true* according to its specification, so we see from the body of *twist* that its execution does not terminate. If it does not terminate, then *halts* returns *false* , and so execution of *twist* terminates. This is a contradiction (inconsistency). Therefore function *halts* cannot have been programmed according to its specification; *halts* is incomputable.

It's really easy to program the body of *halts* . Either it's

```
    begin halts:= true end
```

which returns *true* , or it's

```
    begin halts:= false end
```

which returns *false* . There's no programming problem here. The problem is to decide whether we want *halts* to return *true* or *false* . Whichever one we decide on, it won't satisfy the specification of *halts* . There is a logical inconsistency in the specification of *halts* .

Models of Computation

When we say that a function is “computable”, or that it is “incomputable”, we mean that it can, or cannot, be computed by a Turing-Machine-equivalent computer. All computers currently in use are Turing-Machine-equivalent, or rather, they would be if they had infinite memory, so we usually don't bother to say “Turing-Machine-equivalent”. We can easily build a computer that cannot compute all that a Turing-Machine-equivalent computer can compute: for example, a finite-state automaton. Maybe someone will discover a model of computing that can compute functions that cannot be computed by a Turing-Machine-equivalent computer, although the Church-Turing Thesis says that's not possible.

To prove that a function is (in)computable, the proof must make substantive use of the model of computing with respect to which the function is (in)computable. A proof that *X* is computable could be a program that computes *X* using the instructions of the model. A proof that *X* is incomputable could be a structural induction over all programs of the model.

In any standard proof of incomputability of the halting function, there is an explicit assumption (for the purpose of contradiction) that a halting program has been written, and it is either stated or clearly assumed that this program is written in a Turing-Machine-equivalent programming language, and applies to all programs written in that same language. But nowhere does the proof make substantive use of that assumption. This paper uses Pascal, which is Turing-Machine-equivalent. But suppose we allow *halts* to be an oracle that works by magic; still we come to the contradiction that *halts* ('*twist*', '*twist*') can neither return *true* nor return *false* . That's because the contradiction does not depend on any computing model. It depends only on the specification of *halts* . The specification of *halts* is inconsistent.

Turing defined a model. Turing's proof of incomputability of halting talks vaguely about the “motion” of his machine, just as the above specification of *halts* may be accompanied by a statement saying that it is to be written in Pascal. But the proof makes no use of that model of computation. The proof depends only on what is to be computed, not on how it is to be computed. Any such proof fails to prove incomputability.

Conclusion

If “incomputable” meant having an inconsistent specification, then *halts* would be incomputable. But “incomputable” doesn't mean “inconsistent”. It means that a well-defined function, one with a consistent specification, cannot be computed by a Turing-Machine-equivalent computer, and apply to all programs on that same computer. The *halts* specification is inconsistent; no function satisfies it; so the question of computability does not arise. For a full discussion, see [0].

References

[0] E.C.R.Hehner: Problems with the Halting Problem, *Advances in Computer Science and Engineering* v.10 n.1 p.31-60, 2013 March, hehner.ca/PHP.pdf

[1] A.M.Turing: on Computable Numbers with an Application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society* s.2 v.42 p.230-265, 1936; correction s.2 v.43 p.544-546, 1937

other papers on halting