Halting, the Power of Mathematics, and Religion

Eric C.R. Hehner

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

Halting

The Halting Problem is this: there is a mathematical halting function that says, for each computer program, whether its execution terminates; but there is no computer program to implement this mathematical function. Here is the proof, as first proved by Turing [1], except that I refer to the Pascal programming language instead of the Turing Machine language, and I am using text (character string) parameters instead of numeric parameters.

Let *MathHalt* (p, x) be the mathematical halting function. Parameter p is a text representing a Pascal procedure with one text parameter x.

- *MathHalt* $(p, x) = true$ if execution of the procedure represented by p on input x terminates
- *MathHalt* $(p, x) = false$ if execution of the procedure represented by p on input x does not terminate

Suppose, in order to show a contradiction, that there is a Pascal function *PascalHalt* that implements *MathHalt*. That is: *PascalHalt* $(p, x) =$ *MathHalt* (p, x) for all p and x. We provide a dictionary of Pascal procedure and function definitions so that p can be just the name of a procedure, and then *PascalHalt* can look up the name, and any names of called procedures and functions, and retrieve their definitions for analysis. Now consider this Pascal procedure:

procedure *twist* (x: **string**); **begin if** *PascalHalt* ('*twist*', x) **then** *twist* (x) **end**

Execution of *twist* ('*twist*') begins by calling *PascalHalt* ('*twist*', '*twist*'). If *PascalHalt* ('*twist*', '*twist*') returns *true*, then the execution of *twist* ('*twist*') calls *twist* ('*twist*') and is nonterminating, so *MathHalt* ('*twist*', '*twist*') = *false*, so

PascalHalt ('*twist*', '*twist*') ≠ *MathHalt* ('*twist*', '*twist*')

contrary to the supposition. If *PascalHalt* ('*twist*', '*twist*') returns *false*, then the execution of *twist* ('*twist*') terminates, so *MathHalt* ('*twist*', '*twist*') = *true*, so again

PascalHalt ('*twist*', '*twist*') ≠ *MathHalt* ('*twist*', '*twist*')

contrary to the supposition. Either way, *PascalHalt* does not implement *MathHalt*.

The reason that *MathHalt* cannot be implemented as a program, according to standard theoretical computer science, is the limited power of computation, compared to the (unlimited?) power of mathematics. There is a field of research called hypercomputation that studies computation strengthened by magical powers. There are many journal articles and books on the subject. This field was begun by Turing in [2]; he strengthened the power of computation by adding a magic oracle to determine halting. But, as Turing found out, this still does not solve the problem.

Fortify Pascal with *oracle*, defined such that *oracle* ('*MathHalt*', p, x) = *MathHalt* (p, x), and simultaneously fortify *MathHalt* to apply to fortified-Pascal procedure *twist*, defined as

procedure *twist* (x: **string**); **begin if** *oracle* ('*MathHalt*', '*twist*', x) **then** *twist* (x) **end**

Execution of *twist* ('*twist*') begins by calling *oracle* ('*MathHalt*', '*twist*', '*twist*'). If *oracle* ('*MathHalt*', '*twist*', '*twist*') returns *true*, then the execution of *twist* ('*twist*') calls *twist* ('*twist*') and is nonterminating, so *MathHalt* ('*twist*', '*twist*') = *false*, so

oracle ('*MathHalt*', '*twist*', '*twist*') ≠ *MathHalt* ('*twist*', '*twist*')

contrary to its definition. If *oracle* ('*MathHalt*', '*twist*', '*twist*') returns *false*, then the execution of *twist* ('*twist*') terminates, so *MathHalt* ('*twist*', '*twist*') = *true*, so again

oracle ('*MathHalt*', '*twist*', '*twist*') ≠ *MathHalt* ('*twist*', '*twist*')

contrary to its definition.

There is an inconsistency. If we blame the inconsistency on the specification of *MathHalt*, then there is no mathematical halting function, so we cannot conclude that "the halting function" is incomputable. So it is commonly agreed to blame the inconsistency on the specification of *oracle*. We cannot consistently define, let alone implement, a magic oracle to report on the mathematical halting function.

Let us now repeat the argument, but replacing the mathematical function *MathHalt* with a Python function *PythonHalt*. Here is its header, followed by a comment that specifies the result.

def *PythonHalt* (*p*, *x*):
    """Parameter *p* is a text representing a Pascal procedure with one text parameter *x*.
    PythonHalt (*p*, *x*) = *true* if execution of the procedure represented by *p*
                                on input *x* terminates
    PythonHalt (*p*, *x*) = *false* if execution of the procedure represented by *p*
                                on input *x* does not terminate"""

Thus *PythonHalt* is specified identically to *MathHalt*. Both are outside the set of Pascal procedures they apply to. A Pascal procedure cannot call a Python function, so we translate *PythonHalt* to Pascal. Let *translate* be a Pascal function such that

translate ('*PythonHalt*', *p*, *x*) = *PythonHalt* (*p*, *x*)

Now consider this Pascal procedure:

procedure *twist* (*x*: **string**); **begin if** *translate* ('*PythonHalt*', '*twist*', *x*) **then** *twist* (*x*) **end**

Execution of *twist* ('*twist*') begins by calling *translate* ('*PythonHalt*', '*twist*', '*twist*'). If *translate* ('*PythonHalt*', '*twist*', '*twist*') returns *true*, then the execution of *twist* ('*twist*') calls *twist* ('*twist*') and is nonterminating, so *PythonHalt* ('*twist*', '*twist*') = *false*, so

translate ('*PythonHalt*', '*twist*', '*twist*') ≠ *PythonHalt* ('*twist*', '*twist*')

contrary to its definition. If *translate* ('*PythonHalt*', '*twist*', '*twist*') returns *false*, then the execution of *twist* ('*twist*') terminates, so *PythonHalt* ('*twist*', '*twist*') = *true*, so again

translate ('*PythonHalt*', '*twist*', '*twist*') ≠ *PythonHalt* ('*twist*', '*twist*')

contrary to its definition.

In this argument, *PythonHalt* plays the same role as *MathHalt* played previously, and *translate* plays the same role as *oracle* played. The proof of inconsistency is identical to previously. Whereas previously *oracle*, not *MathHalt*, was blamed, this time, inexplicably, the usual conclusion is opposite to previously: *PythonHalt* is blamed, not *translate*. It is commonly concluded that we cannot program *PythonHalt* because of the limited power of computation, but if we could, we could translate *PythonHalt* into Pascal.

In the next version of the argument, we keep the definition of *PythonHalt*, but instead of translating it to Pascal, we write a Pascal function *interpret* that interprets Python functions, so that *interpret* ('*PythonHalt*', *p*, *x*) = *PythonHalt* (*p*, *x*). Now consider the Pascal procedure

procedure *twist* (*x*: **string**); **begin if** *interpret* ('*PythonHalt*', '*twist*', *x*) **then** *twist* (*x*) **end**

Exactly the same argument as before leads to exactly the same contradiction. As with *translate* and opposite to *oracle*, the usual conclusion is that *PythonHalt*, not *interpret*, is to blame. It is commonly concluded that we cannot program *PythonHalt* because of the limited power of computation, but if we could, we could interpret *PythonHalt* in Pascal.

All of these arguments can be simplified without loss. The parameter *x* played no role in the arguments. It could have been instantiated with any text, and the arguments remain the same. We could leave it out, and talk about the halting status of Pascal procedures that have no parameters, and the arguments remain the same. Since we want to apply *MathHalt* and *PascalHalt* and *PythonHalt* to only one procedure each, we can simplify them by removing parameter *p* and defining them specifically for the one procedure we want to apply them to. Likewise *oracle*, *translate*, and *interpret* can be defined specifically for the one instance they are applied to. All of these parameters serve only to obfuscate. Here are the simplified versions.

Define mathematical binary (boolean) value *MathHalt* as:
- *MathHalt* = *true* if execution of Pascal procedure *twist* terminates
- *MathHalt* = *false* if execution of Pascal procedure *twist* does not terminate

Since execution of *twist* (defined below) either terminates or does not terminate, but not both, *MathHalt* is one of the two binary values.

Suppose, in order to show a contradiction, that there is a Pascal binary value *PascalHalt* that implements *MathHalt*. That is: *PascalHalt* = *MathHalt*. Now define Pascal procedure *twist*:

procedure *twist*; **begin if** *PascalHalt* **then** *twist* **end**

Execution of *twist* begins by evaluating *PascalHalt*. If *PascalHalt* = *true*, then the execution of *twist* calls *twist* and is nonterminating, so *MathHalt* = *false*, so *PascalHalt* ≠ *MathHalt*, contrary to the supposition. If *PascalHalt* = *false*, then the execution of *twist* terminates, so *MathHalt* = *true*, so again *PascalHalt* ≠ *MathHalt*, contrary to the supposition. Either way, *PascalHalt* does not implement *MathHalt*.

We have two equations. The first equation
(0) *PascalHalt* = *MathHalt*
is the definition of *PascalHalt*. The second equation:
(1) *MathHalt* = ¬*PascalHalt*
says that *MathHalt* is the negation of *PascalHalt*. It is obtained by examining *twist*: if *PascalHalt* = *false*, *twist*'s execution terminates, and so, by the definition of *MathHalt*, *MathHalt* = *true*; if *PascalHalt* = *true*, *twist*'s execution does not terminate, and so, by the definition of *MathHalt*, *MathHalt* = *false*. The two equations (0) and (1) are inconsistent.

Is *MathHalt* an incomputable binary value? In other words, is *MathHalt* mathematically well defined to be one of the two binary values, but we are unable to compute which one? And is this because mathematics is powerful enough to define *MathHalt*, but computation is too weak to compute it?

Let us fortify Pascal with a magic binary value *oracle*, defined as *oracle* = *MathHalt*, and simultaneously fortify *MathHalt* to tell us about fortified-Pascal procedure *twist*:
- *MathHalt* = *true* if execution of fortified-Pascal procedure *twist* terminates
- *MathHalt* = *false* if execution of fortified-Pascal procedure *twist* does not terminate

And here is fortified-Pascal procedure *twist* :

procedure *twist*; **begin if** *oracle* **then** *twist* **end**

Now the two equations are:
(2) *oracle* = *MathHalt* (by definition of *oracle*)
(3) *MathHalt* = ¬*oracle* (by examination of *twist* and definition of *MathHalt*)
The inconsistency of (2) and (3) is the same as the inconsistency of (0) and (1). Using magic doesn't help. The standard way out of the inconsistency is to say that there is no magic *oracle* .

We can simplify further by leaving out *oracle* , and just suppose that fortified-Pascal can use the mathematical binary value *MathHalt* directly. Procedure *twist* becomes:

procedure *twist*; **begin if** *MathHalt* **then** *twist* **end**

Now we have only one equation:
(4) *MathHalt* = ¬*MathHalt* (by examination of *twist* and definition of *MathHalt*)
Equation (4) is inconsistent all by itself, and we have no *oracle* to blame. We might blame the fact that we cannot use a mathematically defined name in a program. But in the definition of *twist* , can't we replace the name *MathHalt* with its value, *true* or *false* , which is part of Pascal? We don't need to fortify Pascal with magic or with mathematics, and we don't need to suppose there is a Pascal function *PascalHalt* , in order to arrive at the same inconsistency.

Replacing mathematics with Python, but leaving out the obfuscating parameters, define:

def *PythonHalt*: """*PythonHalt* = *true* if execution of the Pascal procedure *twist* terminates
PythonHalt = *false* if execution of the Pascal procedure *twist*
does not terminate"""

Let *translate* be the Pascal binary value, *true* or *false* , that translates the Python binary value *PythonHalt* into Pascal: *translate* = *PythonHalt* . Define:

procedure *twist*; **begin if** *translate* **then** *twist* **end**

These definitions are the two equations:
(5) *translate* = *PythonHalt* (by definition of *translate*)
(6) *PythonHalt* = ¬*translate* (by examination of *twist* and definition of *PythonHalt*)

If you think there is a difference between translation and interpretation of a binary value, then let *interpret* be the Pascal interpretation of *PythonHalt* . That is: *interpret* = *PythonHalt* . Define:

procedure *twist*; **begin if** *interpret* **then** *twist* **end**

These definitions are the two equations:
(7) *interpret* = *PythonHalt* (by definition of *interpret*)
(8) *PythonHalt* = ¬*interpret* (by examination of *twist* and definition of *PythonHalt*)
We have the same inconsistency.

It makes no difference to the inconsistency argument whether we have a mathematically defined binary value (*MathHalt*) or a program-defined binary value (*PythonHalt*). All that matters is that the binary value be defined outside Pascal. And it makes no difference to the inconsistency argument whether we use magic (*oracle*) or a program (*translate* , *interpret*) to represent, inside Pascal, that binary value defined outside Pascal.

Mathematics and Religion

There are two kinds of mathematician: idealists and realists. Idealist (usually called Platonist) mathematicians are like religious people, and realist (often called formalist) mathematicians are like atheists. Idealists believe that mathematical objects, like numbers and sets and functions, exist, even if you can't see them or hear them or sense them in any way; religious people believe that god(s) and souls exist, even if you can't see them or hear them or sense them in any way. Idealist mathematicians believe there are mathematical truths, which they discover; religious people believe there are religious truths, which are revealed to them. When idealists are confronted by an inconsistency between a mathematical function (*MathHalt*) and a computational function (*PascalHalt*), they maintain faith in the existence of the mathematical function, which they believe is not a human creation, and deny the existence of the computational function, which clearly would be a human creation. Idealist mathematicians believe that mathematics is all-powerful, and computation has limited power, just as religious people believe that God is all-powerful, and humans have limited power.

An atheist denies the existence of gods and souls, and a realist mathematician denies the existence of mathematical objects. To realists, mathematics is a language, and numbers, sets, functions, and so on, are expressions in the language of mathematics. You can see them: ink on paper, or pixels on a screen. A realist mathematician believes that mathematics is a human creation. Applied mathematics is invented for the purpose of modeling and reasoning about real objects and phenomena. We can judge a mathematical creation by how well it models and aids understanding and helps us to reason about reality. There are no mathematical truths, but mathematical expressions can be used to represent real truths. (It is unclear why pure mathematics is invented, and unclear what guides its invention. It is sometimes justified by saying that it may find applications later.)

To a realist, mathematics is a language not very different from a programming language. There are rules to say which sequences of symbols are mathematical expressions and which are not, and a computer can check a sequence of symbols for syntactic correctness, just as it does for programs. (The rules change from context to context, and they change over time with each new mathematical invention, just as they do from programming language to programming language. But in any given context, at any given moment, there are rules.) There are axioms and proof rules to say which mathematical expressions are theorems, and a computer can check a proof for correctness. An idealist may imagine that their mathematical expressions are talking about a rich world of mathematical objects, but the expressions are subject to syntax checks and proof checks, and those are computational. To a realist, there is no reason to think that mathematics is more powerful than computation (whatever that might mean). If there is a difference between the power of mathematics and the power of computation, that difference was not used anywhere in the halting problem inconsistency arguments, and so it is not the reason for the inconsistency.

An idealist says that a mathematical object exists exactly when a realist says that some mathematical expressions are consistent. To the idealist, those expressions are a specification of the object. Idealist existence is realist consistency. The only way to prove that the mathematical halting function is consistently defined (exists) is to implement it.

> [long aside] Architects, physicists, economists, civil engineers, neuroscientists, computer scientists, climate modelers, and many other people, use models. To all of them, a model is something simpler and more abstract than what it models. An architect may create a balsa wood and paint model that shows the shape and color of a building, but not what size it is, nor how much it weighs. The architect may create a blueprint model that states the dimensions of the building but not its weight or color. An actual building is an implementation of an architect's model; it is more detailed than the model; it has a shape and size and weight and color.

> A mathematical theory is a model. For example, Newton's theory of motion allows us to calculate the trajectory of a cannonball, but does not tell us what color the cannonball is. A flying cannonball is an implementation of the theory; in addition to its trajectory, it has a color. Computer scientists build a model to try to understand some software; to be useful, the model has to be simpler than the software; the software implements the model. Theories and specifications are models of reality.
>
> Logicians are a peculiar kind of mathematician. They use the word "model" with the opposite meaning from everyone else. To a logician, a model is more detailed, less abstract, than the theory it models. Logicians use the word "model" to mean implementation. Their preferred implementation language is set theory. I think logicians have done a great disservice, sowing confusion, by misusing the word "model". [end of long aside]

The only way to prove that the mathematical halting function is consistently defined (exists) is to implement it. The mathematical function that expresses halting of Pascal programs cannot be implemented in Pascal; that's the Halting Problem. But perhaps it can be implemented in Python (or any other programming language outside Pascal). If so, that Python (or other language) program cannot be translated to (or interpreted in) Pascal. If we fail to implement the mathematical halting specification (as a program or in set theory), then we do not know that the mathematical specification is consistent; we do not know that there is a mathematical halting function. If we succeed to implement it as a program, then the specification (definition) of halting is consistent, there is such a function, and it is computable.

In 1931, Kurt Gödel proved [0] that the mathematics of Russell and Whitehead's *Principia Mathematica* [3] (*PM*) is incomplete, assuming it is consistent. To do so, he defined in *PM* a function (*Bew*) that, applied to any encoded sentence of *PM*, says whether that sentence is a theorem of *PM*. In essence, Gödel used *PM* as a programming language, writing an interpreter for *PM*. The program was written in a style that we call "functional" today. It wasn't recognizable then, but Gödel was implementing mathematics as a computer program. We now know:

- All of mathematics can be ~~modeled~~ implemented in standard set theory (for example, Zermelo-Fraenkel set theory).
- Set theory can be implemented in a programming language (for example, any automated prover).
- Therefore all of mathematics can be implemented in a programming language.

This implies that mathematics is Turing-Machine-Equivalent. Mathematics is not, in any sense, more powerful than computing.

Character Judge

The following story is intended as an analogy to the Halting Problem.

A person might admire another person, or not. A person might admire themself, in which case they are conceited, or they might not admire themself, in which case they are modest. A character judge for a town admires the modest people of the town, and does not admire the conceited people of the town. There is a town named Pascalville, and in it there is a person named Blaise. Blaise cannot be a character judge for Pascalville because, if he admires himself, he is conceited, and a character judge does not admire a conceited person, and if he does not admire himself, he is modest, and a character judge admires a modest person.

Outside Pascalville there are two other communities. One is Heaven, where God lives, and the other is Pythonville, where Monty lives. God is a character judge for Pascalville; he uses his infinite wisdom to look deep into the souls of the people of Pascalville. Monty is also a

character judge for Pascalville; he is only human, but he is intelligent and a good judge of character. It is just as consistent to say that Monty is a character judge for Pascalville as it is to say that God is a character judge for Pascalville. The only attribute required for consistency is that they do not live in Pascalville.

Blaise is unhappy; he can't see why both God and Monty can be character judges for Pascalville, but he cannot. Blaise prays to God, and God answers, telling Blaise whom he admires. So Blaise decides to do the same: whomsoever God admires, so shall Blaise admire. But, try as he might, Blaise cannot do the same as God. The reason is that God admires Blaise if and only if Blaise does not admire Blaise. But Blaise is baffled. He thinks the reason he cannot do the same as God might be that he is not hearing God correctly. Most people believe that Blaise cannot do the same as God because God is all powerful, and Blaise is not.

Blaise tries again. This time he talks to Monty. Monty tells Blaise whom he admires. Blaise decides to do the same as Monty, admiring whoever Monty admires, not admiring whoever Monty does not admire. But Blaise is still not a character judge for Pascalville. The reason is the same as before: Monty admires Blaise if and only if Blaise does not admire Blaise. But Blaise is still baffled. He is sure he hears Monty correctly. Most people believe that Blaise is doing the same as Monty, but Monty isn't really a character judge for Pascalville.

The analogy to the Halting Problem is as follows:
- person: program
- Pascalville: the Pascal programming language
- person who lives in Pascalville: program in the Pascal programming language
- modest person: program whose execution halts
- conceited person: program whose execution does not halt
- character judge for a town: halting function for a programming language
- Blaise: an arbitrary Pascal program
- Heaven: mathematics
- God as character judge for Pascalville: the mathematical halting function for Pascal programs
- Pythonville: the Python programming language
- Monty as character judge for Pascalville: a Python program to compute halting for Pascal programs
- infinite wisdom: the unlimited power of mathematics (or, at least, the greater power of mathematics than the power of computation)
- soul of a person: text of a program
- human: the limited power of computation
- God speaks to Blaise: an oracle, as introduced by Turing
- Blaise does not hear God correctly: there is no such thing as an oracle
- most people believe: computer science textbooks say
- Blaise does the same as Monty: a Pascal program that is the translation of a Python program, or a Pascal program that interprets a Python program; a Pascal program whose execution does the same as the execution of a Python program
- Monty isn't really a character judge for Pascalville: there is no Python program to compute halting for Pascal programs

Conclusion

The story is a faithful analogy. I am hoping the analogy makes clear that what “most people believe” is unwarranted. If we believe that the mathematical halting function exists (has a consistent specification), we can prove that it cannot be implemented in the programming language to which it applies, but we cannot conclude that it cannot be implemented in another programming language, so we cannot conclude that it is incomputable.

References

[0] K.Gödel: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I, *Monatshefte für Mathematik und Physik* v.38 p.173-198, Leipzig, 1931.

[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

[2] A.M.Turing: Systems of Logic based on Ordinals, p.8, Ph.D. thesis, Princeton University, 1939

[3] A.N.Whitehead, B.Russell: *Principia Mathematica* volumes 1, 2, and 3, Cambridge University Press, 1910, 1912, 1913

This paper was rejected by arXiv.org cs.LO on 2021-6-15.

Appendix added 2021-6-24

I have used the word “idealist” to describe a mathematician who believes that mathematical objects exist, and the word “realist” to describe a mathematician who believes that there are no mathematical objects, only mathematical expressions. Strangely, other people have used the word “realist” where I have used the word “idealist”, and “anti-realist” where I have used the word “realist”. It seems to me that each kind of mathematician, those who believe mathematical objects exist and those who don't, want to say they are the “realists”, because that makes their position sound like the right one.

other papers on halting