Possible interpretations of the halting problem

Zoltán Galántai PhD
ELTE GTI (Hungary)

"The opposite of a correct statement is a false statement. But the opposite of a profound truth may well be another profound truth." — Niels Bohr

Abstract

Turing originally examined the halting problem to find an answer for Hilbert's Entscheidungsproblem and his conclusion was roughly that since it is impossible to decide whether a program would halt or run forever, there is no solution for Hilbert's problem. The typical argumentation for this undecidability is based on a contradiction of a self-referential logical paradox. Our aim is to point out that contrary to the widely accepted opinion, even if we accept the existence of the paradox, it can be stated that thanks to the new results of philosophy of the science that has appeared since the publication of Turing's work there is more than one possible interpretation for the halting problem and there is more than one possible implementation if we want to write a program. Thus it is not impossible to decide whether an arbitrary program would halt within finitely many steps if we accept certain starting points.

Introduction

Turing published his paper entitled "On Computable Numbers, with an Application to the Entscheidungsproblem" in 1937 and his proof for the so-called halting problem is one of the essential thesis of the computability theory. According to this, it is impossible to decide about an arbitrary program whether it would stop or would run forever [Turing 1937]. We have to emphasize that Turing worked in the era of logical positivism when–necessarily oversimplifying this approach–it was widely believed that science is about "truth" and that there are no acceptable (alternative) solutions. Today it is accepted in the philosophy of science that, instead of searching for pure "facts" or "truths", we can examine different interpretations [Babbie 2012].

Similarly, It would be a methodological mistake (an anachronism) to forget that Turing's work is a result of a certain era's interpretation of mathematical problems, and there was no programming at that time. So it is a question how much the original answer can be applied to real software.

One of the popular demonstrations of Turing's solution for the halting problem is based on an imagined machine (a program / software) that can make certain decisions. A machine's state as output determines our machine's input and if the source of both the output and the input is the same, then the result can be an unsolvable paradox.

In the following, we would like to point out that even if the original mathematical paradox is unanswerable, an existing program always and necessarily will either run forever or will stop in maximum two steps–supposing that the result is determined solely by our decision to handle the problem and we are to choose the simplest possible solutions. Obviously, it is possible to write programs that would stop not in one or two, but *n* steps. The point is that its behavior is totally predictable if we know the program code.

This is the result of the program code and thus the results are unavoidably influenced by the programmer's decision about the handling a of the problem. And both the outputs (run forever / halt) are equal for a certain situation since they are caused by the programmer's decision and neither solution is more solid or weaker than the other. (Notice that this discussion is only about the question whether a program would halt because of its logic. This

is not applicable, for example, to the Collatz conjecture where we ask whether a certain sequence would halt within finitely many steps: This problem is not caused by the structure of the code.)
At this point, we have to mention that the role of the programmer's decisions raises the question whether there are cases, regarding theorem proving by computers, when a similar programming decision changes the result.

The Entscheidungsproblem and the halting problem

Truing examined the computable numbers that are "real numbers whose expressions as a decimal are calculable by finite means... a number is computable if its decimal can be written down by a machine" and he stated that these numbers are only a subset of the definable numbers. His main conclusion is that "the Hilbertian Entscheidungsproblem can have no solution" [Turing 1937].
The Entscheidungsproblem was formulated by David Hilbert in 1928 asking whether there is an algorithm that using a statement of a first-order logic as an input is able to answer "yes" if the statement's validity is universal and answers "no" if the statement is not universally valid. In other words: the question is whether there is an algorithm to decide if a certain mathematical assertion has a proof [Cooper 2017].
If there is an undecidable question, then Hilbert's universal decision algorithm is not possible and if Turing's halting problem (as an analogy for the Entscheidungsproblem) is unanswerable because of a logical contradiction, then this is the case.
Obviously, even if Turing's proof for the halting problem is problematic, it is possible that there is no algorithm to answer the Entscheidungsproblem –if we have another proof for its impossibility.

Modelling the halting problem

Imagine a machine (code) that can detect whether a machine's output is "yes" or "no". These are the only possible states of the machine; and "yes" would mean that the machine runs infinitely while "no" means that it stops in a finite number of steps. Ignoring the technical details, such as the Turing completeness, it is enough to mention that we can prescribe to the machine to answer for a "yes" output a "yes" and for a "no" a "no". Namely, if the examined program stops then this examining program stops (or if the examined program runs, then it runs) as well. Similarly, we can choose other rules prescribing an inverse answer where the examining program's answer is the opposite of the examined one's ("yes" for "no" and "no" for "yes").

Table 1. Possible* if - then *relations between the outputs of the machines
Only the first two variations are examined in connection with the halting problem.

If "yes" then "yes" If "no", then "no"	result is either "yes" or "no" (no logical contradiction)
If "yes" then "no" If "no", then "yes"	logical contradiction (see Entscheidungsproblem)
If "yes" then "no" If "no", then "no"	result is necessarily "no" (no logical contradiction)
If "yes" then "yes" If "no", then "yes"	result is necessarily "no" (no logical contradiction)

But this logical construction becomes problematic if a machine examines itself since the result is a logical contradiction: If the state of the machines is “yes”, then it is “no” and if it is “no”, then it is “yes”. So the real state of the machine is undecidable [MacCormick 2012]. Of course, even if we believe that this contradiction is unsolvable logically, we–contrary to a popular interpretation–do not have to assume that the result is an infinite loop. After all, if it is unsolvable, then using the rules of logic we cannot say what the result is–and a loop would be a result.

Barbers, R-sets, and paradoxes

The halting problem seems to be similar to other, well–known paradoxes (e.g. the barber paradox). But the self–referential nature does not make automatically identical the different contradictions [Sainsbury 2009]. For example, the barber paradox and Russell’s R-sets are similar regarding their logical structure, but they are different from a certain point of view.
In the first case, the question is who shaves the barber in a city where everyone (every man) is shaved and nobody is shaved by himself. In the second case the question is the following: a set is called abnormal if it contains itself and normal if not. So what’s about the set of all the normal sets (R)? Does it contain itself?
“The difference is this: nothing leads us to suppose that there is such a barber; but we seem to be committed, by our understanding of what it is to be a class, to the existence of R” and this causes a paradox [Sainsbury 2009]. To give an example: the barber paradox can be solved by introducing two barbers but this is not possible in the case of the R sets.
Ad analogiam: it can be argued that the halting problem is not equal to the paradox that is the core of the Entscheidungsproblem, since even if we accept the undecidability of the state of the machine (that it is impossible to decide whether it is “yes” or “no”), that does not mean that it is impossible to decide whether the machine would halt under certain conditions.
In our case the main difference between the logical question and the machine (program) is that in the second case if we want to write a real program, we have to decide how to handle the logical contradiction. And since there is no clear and unquestionable solution for handling it, we have to choose one.
Working on the halting problem in a pre-programming era, Turing did not write a real code: He interpreted the problem as a purely mathematical one that can be examined by constructing logical machines. But the logic represented by these machines and the logic of the problem–solving mechanism of the machines can be different.

Halting problem vs. dividing by zero

I do not want to say that it is an unquestionable solution to construct a program that contains a mathematical contradiction (paradox), but it is similarly not unquestionable to refuse the program writing arguing that because of the contradiction it is pointless (this is an unspoken assumption of those who accept Turing’s solution). But despite the paradox, it is not only possible to produce a program that behaves unequivocally: A program necessarily will give a clear answer.
To construct the paradox, it is enough to choose a reverse rule (if “yes”, then “no” and vice versa) from the set of possible rules (see table below). To write a real program, the second necessary step is choosing an answer for the contradiction on the level of the program code and to decide what output is assigned to it.
If we decide that the paradox is impermissible then we have either to cancel the program run or we have to declare that the paradox causes an infinite loop and thus the program never halts. Obviously, both of these solutions are arbitrary in the sense that there is no stronger argument to support the first than the second solution.

But, in parallel, it can be clearly decided in both cases whether the program will halt or not (although, as it was underlined earlier, it does not mean that we can solve the paradox).
The classic example of the first solution in programming is handling the division by zero problem. E.g. in Python the answer is that "ZeroDivisionError: division by zero" and this causes to halt the program.
To summarize this approach: If we have a logical contradiction then we have to halt since the results after the contradiction are meaningless. The halting of the program does not make interpretable the result of the division by zero–but this gives an answer for the problem not at the level of either logic or mathematics, but at the level of programming.

In pseudo code:

```
IF contradiction then:
    halt
ELSE:
    run
ENDIF
```

(At this point we ignore the possibility that the program can halt even if it works normally (e.g. because it finishes the calculation) since the question is the impact of the contradiction on the running. After all, according to the original interpretation of the halting program, this makes unsolvable the problem but the regular working of the program is not problematic.)
Similar to the division by zero problem, in the case of the halting problem, it seems to be acceptable to argue that there is a contradiction between the "yes" and "no" answers and because of that, the program should halt.
But it is similarly acceptable to argue that the contradiction causes an infinite loop (if the output is "yes" then the output is "no"; if "no" then "yes" etc.) and thus the program never halts.

In pseudo code:

```
IF contradiction then:
    run
ELSE:
    halt
ENDIF
```

As it can be seen, it depends on the programmers decision whether the program would halt or run forever in the presence of the logical contradiction and from our point of view, it is no question what the output is ("yes" or "no"), but whether the program halt.

Yet another solution for the halting problem: sequential approach

According to Eric C. R. Hehner, in connection with the halting problem, we have to choose between incompleteness and inconsistency. If we do not decide the program's response (and thus we accept incompleteness), then it is per definitionem impossible to get an answer about the program's behavior and thus it is a logical error to "conclude that it is incomputable". And "[I]f we choose inconsistency, then it makes no sense to propose a halting program" [Hehner 2013].
Hehner's argumentation based on the traditional approach focusing on the logical inconsistencies of the halting problem. But we can choose to question the validity of the core paradox of the halting problem itself. In Turing's approach the definition of the computation is based on an abstract mathematical description and because of that, the solution of an equation is necessarily a result of a completed process.

But we can interpret this situation as a process in time. We can require the program to examine the state of the output in the first step then to follow the rule assigned to it. This means that according to the traditional approach if the output is “yes” this causes the same paradox as if the output is “no”. However, according to a possible, sequential interpretation, the “yes” output requires to halt and the “no” requires to run. Thus if the actual output is “yes” then the program halts immediately and in the case of “no” it runs. Of course, we have to decide whether the program starts with “yes” or “no” state; but such a decision is necessary since a parameter (the output) is uninterpretable without a value assigned to it in programming.

The two possible results:
Beginning state of output: “yes”:

- if “yes” then “no” thus the program halts.

Beginning state of output: “no”:

- if “no” then “yes” thus the program runs.
- if “yes” then “no” thus the program halts.

In pseudo code:

```
WHILE program runs:
    IF output = “yes” then:
        output = “no”
        halt
    IF output = “no” then:
        output = “yes”
        run
ENDWHILE
```

In sum, besides the above–mentioned solutions (where the contradiction causes either a prompt halt or an infinite run of the program) there is a third one. If we accept that the “yes” and “no” outputs exist not parallel but sequentially then there is no possibility for the program to run forever: Independently from the beginning state of the output, the program necessarily halts within maximum two steps.

Conclusions

As we have shown there are different but equally acceptable (or refutable) answers for the halting problem.

1. The halting problem as a logical problem:

(1a) We can argue that the halting problem is a logical contradiction and thus it is unsolvable. This is the most widely accepted approach today but notice that if we agree with it, then we cannot use the emergence of the infinite loop as a proof of the existence of the paradox since this is not the result of the contradiction. It is the result only of our decision about handling the paradox, and it would be an equivalently acceptable to choose to halt the program immediately (as in the case of the division with zero).
(1b) We can adapt the dialetheism’s approach to this problem (although it has historical antecedents, its emergence is a relatively new development in philosophy). According to it, there is no contradiction between the “yes” and “no” outputs: both of them are true [Priest et all 2018].
These two approaches, (1a) and (1b) mutually exclude each other. At this point we have to recognize that even if we cannot regard the “yes” / “no” problem as a contradiction

(paradox), we have to decide how to handle it at the program level (namely whether it should causes a forever run or a halt) if we want to write a real code.
It is also important to recognize that it depends solely on our decision which can be accepted as "true" (or, at least, as acceptable). While Turing and the founding fathers of modern computer science regarded the core problem of the halting problem as a contradiction that causes a paradox, even this is a matter of interpretation (see dialetheism).

2. The halting problem as a programming problem:

(2a) Accepting either *(1a)* or *(1b)*, we can prefer either the forever run of the program or the prompt halt: For the reasons mentioned above this depends solely on our decision.
(2b) In a similar way, we can argue that it is possible to interpret this question from a sequential point of view and thus we do not have to accept the existence of the contradiction – contrary to the earlier mentioned approaches. In this case it is not indifferent whether the first output is "yes" or "no" but both starting states lead to the halt of the code within one or two steps.
All these solutions are equal in the sense that since there is no method to compare them, we cannot argue that one of them is better than the others. Thus we have to accept that in accordance with certain programming decisions, an arbitrary code will run either forever or will halt soon, but it is always totally foreseeable and from this point of view the halting problem does not exist.

07. Sept. 2018
modified on 08. Sept. 2018

Sources

- Babbie, Earl, 2012: The Practice of Social Research. Wadsworth Publishing
- Cooper, Barry S., 2017: Embodying Computation at Higher Types. In Floyd, Juliet – Bokulich, Alisha, (eds.): Philosophical Explorations of the Legacy of Alan Turing: Turing 100. Springer
- Hehner, Eric R.: Problems with the Halting Problem, COMPUTING2011 Symposium on 75 years of Turing Machine and Lambda-Calculus, Karlsruhe Germany, invited, 2011 October 20-21; Advances in Computer Science and Engineering v.10 n.1 p.31-60, 2013 http://www.cs.toronto.edu/~hehner/PHP.pdf
- MacCormick, John 2011: Nine Algorithms That Changed the Future. THE INGENIOUS IDEAS THAT DRIVE TODAY'S COMPUTERS. Princeton Univ. Press
- Priest, Graham–Berto, Francesco–Weber, Zach, 2018: Dialetheism. In Stanford Encyclopedia of Philosophy https://plato.stanford.edu/entries/dialetheism/
- Sainsbury, R. M., 2009: Paradoxes. Cambridge Univ. Press
- Turing, A. M, 1937: "On Computable Numbers, with an Application to the Entscheidungsproblem"
  https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf