Did Turing ever halt?

Joel David Hamkins
O'Hara Professor of Logic
University of Notre Dame

History and Philosophy of Science Colloquium
University of Notre Dame
17 October 2025

This talk is based on joint work:

[HN24] Joel David Hamkins and Theodor Nenu, “Did Turing prove the undecidability of the halting problem?”, 18 pages, 2024, Mathematics arXiv:2407.00680. Under review.

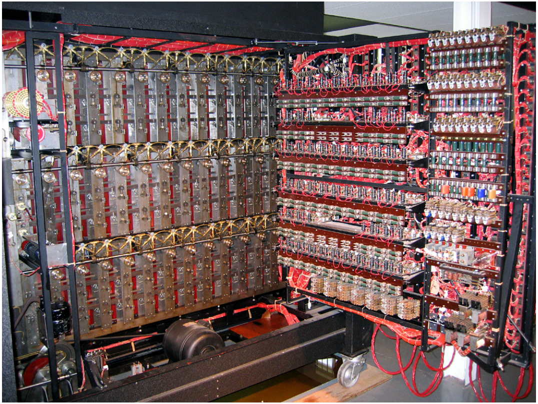
Alan Turing (1912–1954)

Sculpture by Stephen Kettle, Bletchley Park

Enigma

German ‘Enigma’ encryption device

The Bombe



Bombe device, Bletchley Park

On computable numbers, 1936

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable. In § 8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel†. These results

† Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I", *Monatshefte Math. Phys.*, 38 (1931), 173–198.

Turing's 1936 paper, "On computable numbers..."

It is difficult to overstate the importance of Turing's paper, written while he was a student at Cambridge.

Introduces profound, fundamental ideas on computability.

- Introduces Turing's formal conception of computability
- Defines Turing machines
- Proves existence of universal computers
- Identifies undecidability phenomenon
- Solves the Entscheidungsproblem
- Introduces the computable numbers

Turing laid the theoretical foundation for the computer age. One of the most impactful papers ever written.

Turing's concept of computability

Turing reflected philosophically on the nature of computation.

Turing's concept of computability

Turing reflected philosophically on the nature of computation.

There were indeed many computers in that era, and some firms had whole rooms full of computers, the "computer room."

Turing's concept of computability

Turing reflected philosophically on the nature of computation.

There were indeed many computers in that era, and some firms had whole rooms full of computers, the "computer room."

But of course, such a "computer" is not a machine, but a person—more specifically, an occupation. The computer rooms were filled with people hired as computers and tasked with various computational duties, often in finance or engineering.

In old photos, you can see the computers—mostly women—sitting at big wooden desks, with pencils and a sufficient supply of paper. They would perform their computations by writing on the paper, of course, according to various definite computational procedures.

What ‘computers’ do

Turing aimed to model an idealized form of computational processes.

He reflected on the behavior of ‘computers’ (that is, people working as computers) when working on a computational task. They make various marks on paper, depending in part on what they see before them, or after looking at earlier marks.

Eventually, some marks may be indicated as output information.

Nature of computation

Turing realized that some simplifying assumptions do not fundamental affect the nature of computation.

- We may assume marks appear in a grid of cells.
- May assume each cell has only 0 or 1.

Nature of computation

Turing realized that some simplifying assumptions do not fundamental affect the nature of computation.

- We may assume marks appear in a grid of cells.
- May assume each cell has only 0 or 1.
- Two dimensions not important; assume a line of cells.
- May assume computer has limited memory, but as much paper as needed.

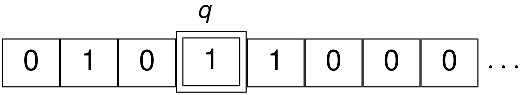
Nature of computation

Turing realized that some simplifying assumptions do not fundamental affect the nature of computation.

- We may assume marks appear in a grid of cells.
- May assume each cell has only 0 or 1.
- Two dimensions not important; assume a line of cells.
- May assume computer has limited memory, but as much paper as needed.
- Rudimentary actions determined by current ‘state’ of mind.

Turing machines

Thus, Turing derived his machine concept of computation.

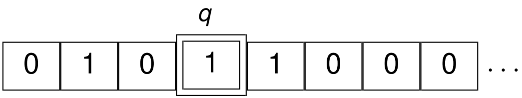


A Turing machine has an infinite paper tape, divided into cells, which accommodate 0 and 1.

The head, at any moment in one of finitely many *states*, reads and writes on the tape, moving according to the rigid instructions of a program.

Turing machines

Thus, Turing derived his machine concept of computation.



A Turing machine has an infinite paper tape, divided into cells, which accommodate 0 and 1.

The head, at any moment in one of finitely many *states*, reads and writes on the tape, moving according to the rigid instructions of a program.

$$(q, a) \mapsto (r, b, d).$$

When in state q reading symbol a, then change to state r, write symbol b, and move one cell in direction d.

Computable numbers

Turing introduces the concept of *computable numbers*.

Computable numbers

Turing introduces the concept of *computable numbers*.

He defines that a real number is *computable*, when there is a computational procedure to enumerate the decimal digits of the number.

$$3.14159265358979323846264338327950288419\cdots$$

He proceeds to develop the theory of computable real numbers.

Arithmetic

Consider the ordinary arithmetic operations $a + b = c$.

Arithmetic

Consider the ordinary arithmetic operations $a + b = c$.

$$\begin{array}{rl} & 0.111111111111\ldots \\ + & 0.222222222222\ldots \\ \hline & 0.333333333333\ldots \end{array}$$

We would like to compute the sum of two given numbers.

Arithmetic is more difficult than you expect!

But hang on. Consider the following case of $a + b$.

$$\begin{array}{rl} & 0.333333333333\ldots \\ + & 0.666666666666\ldots \\ \hline & 0.999999999999\ldots \end{array}$$

We can start writing down the answer 0.99999 . . ., but these digits will be wrong if we ever find a carry term, since then the answer should be 1.000000 . . .

Arithmetic is more difficult than you expect!

But hang on. Consider the following case of $a + b$.

$$\begin{array}{rl} & 0.333333333333\ldots \\ + & 0.666666666666\ldots \\ \hline & 0.999999999999\ldots \end{array}$$

We can start writing down the answer $0.99999\ldots$, but these digits will be wrong if we ever find a carry term, since then the answer should be $1.000000\ldots$

But $1.0000000\ldots$ will be wrong if we ever find a digit place with sum less than 9.

Arithmetic is more difficult than you expect!

But hang on. Consider the following case of $a + b$.

$$\begin{array}{rl} & 0.333333333333\ldots \\ + & 0.666666666666\ldots \\ \hline & 0.999999999999\ldots \end{array}$$

We can start writing down the answer $0.99999\ldots$, but these digits will be wrong if we ever find a carry term, since then the answer should be $1.000000\ldots$

But $1.0000000\ldots$ will be wrong if we ever find a digit place with sum less than 9.

It seems that we cannot start to give the digits of $a + b$, given the initial digits of a and b.

Arithmetic on computable numbers

This is actually a fundamental problem.

Arithmetic on computable numbers

This is actually a fundamental problem.

There is no computable procedure to compute the digits of $a + b$, given programs for computing the digits of a and b separately.

Arithmetic on computable numbers

This is actually a fundamental problem.

There is no computable procedure to compute the digits of $a + b$, given programs for computing the digits of a and b separately.

In this sense, ordinary arithmetic is not a computable operation with the digits-conception of computable number.

Arithmetic on computable numbers

This is actually a fundamental problem.

There is no computable procedure to compute the digits of $a + b$, given programs for computing the digits of a and b separately.

In this sense, ordinary arithmetic is not a computable operation with the digits-conception of computable number.

But if one modifies Turing's concept slightly, however, then everything works great!

A solution

Namely, logicians today define that a *computable real number* is a program that computes rational approximations to a real number, as accurately as desired.

As approximations, 0.999999999 and 1.00000000 are very close, even though their digits are totally different.

A solution

Namely, logicians today define that a *computable real number* is a program that computes rational approximations to a real number, as accurately as desired.

As approximations, 0.999999999 and 1.00000000 are very close, even though their digits are totally different.

With this modified concept, all the usual operations are computable.

$$a + b \qquad ab \qquad \sin(x) \qquad e^x$$

Turing's computable real number idea turns into the robust subject known as *computable analysis*.

Decidability

A decision problem is *computably decidable*, if there is a computational method to determine whether the yes/no answer on any given input.

Decidability

A decision problem is *computably decidable*, if there is a computational method to determine whether the yes/no answer on any given input.

A decision problem is *computably enumerable*, in contrast, if there is a computable algorithm that enumerates all and only the positive instances.

Decidability

A decision problem is *computably decidable*, if there is a computational method to determine whether the yes/no answer on any given input.

A decision problem is *computably enumerable*, in contrast, if there is a computable algorithm that enumerates all and only the positive instances.

Question

Are these the same?

Decidability

A decision problem is *computably decidable*, if there is a computational method to determine whether the yes/no answer on any given input.

A decision problem is *computably enumerable*, in contrast, if there is a computable algorithm that enumerates all and only the positive instances.

Question

Are these the same?

It turns out, using extensions of Turing's arguments, that they are different.

Puzzling example

Consider the decision problem: on input *n*, decide whether there are *n* consecutive 8s in the decimal expansion of π.

$$3.14159265358979323846264338327950288419\cdots$$

Puzzling example

Consider the decision problem: on input n, decide whether there are n consecutive 8s in the decimal expansion of π.

$$3.14159265358979323846264338327950288419\cdots$$

Question

Is this decision problem computably decidable?

Puzzling example

Consider the decision problem: on input *n*, decide whether there are *n* consecutive 8s in the decimal expansion of π.

$$3.14159265358979323846264338327950288419\cdots$$

Question

Is this decision problem computably decidable?

Naive attempt: on input *n*, begin to search the decimal expansion of π. If you find *n* consecutive 8s, then say Yes.

Puzzling example

Consider the decision problem: on input n, decide whether there are n consecutive 8s in the decimal expansion of π.

$$3.14159265358979323846264338327950288419\cdots$$

Question

Is this decision problem computably decidable?

Naive attempt: on input n, begin to search the decimal expansion of π. If you find n consecutive 8s, then say Yes.

...but when shall we say No?

More sophisticated answer

Decision problem: are there n consecutive 8s in π?

More sophisticated answer

Decision problem: are there *n* consecutive 8s in π?

This is computably decidable.

More sophisticated answer

Decision problem: are there n consecutive 8s in π?

This is computably decidable.

We argue by cases, depending on the nature of mathematical reality.

More sophisticated answer

Decision problem: are there *n* consecutive 8s in π?

This is computably decidable.

We argue by cases, depending on the nature of mathematical reality.

Case 1. If the fact of the matter is that there are arbitrarily long blocks of 8s appearing in π, then the answer will always be Yes. And this is computable: just say Yes immediately.

More sophisticated answer

Decision problem: are there *n* consecutive 8s in π?

This is computably decidable.

We argue by cases, depending on the nature of mathematical reality.

Case 1. If the fact of the matter is that there are arbitrarily long blocks of 8s appearing in π, then the answer will always be Yes. And this is computable: just say Yes immediately.

Case 2. Otherwise, there is some longest string of 8s, of some length N. But now the answer is Yes if $n \leq N$ and otherwise No. For the particular (unknown but fixed) N, this also is computable.

More sophisticated answer

Decision problem: are there *n* consecutive 8s in π?

This is computably decidable.

We argue by cases, depending on the nature of mathematical reality.

Case 1. If the fact of the matter is that there are arbitrarily long blocks of 8s appearing in π, then the answer will always be Yes. And this is computable: just say Yes immediately.

Case 2. Otherwise, there is some longest string of 8s, of some length N. But now the answer is Yes if $n \leq N$ and otherwise No. For the particular (unknown but fixed) N, this also is computable.

So in any case the problem is computable. We just don't know which algorithm works.

Undecidability

Can there be *undecidable* decision problems?

Undecidability

Can there be *undecidable* decision problems?

Is every mathematical question in principle answerable by a computational procedure?

Undecidability

Can there be *undecidable* decision problems?

Is every mathematical question in principle answerable by a computational procedure?

Turing sought to answer these questions.

Halting problem

Consider the famous *halting problem*: given a program p and input, determine whether it will halt.

Is this computably decidable?

Halting problem

Consider the famous *halting problem*: given a program p and input, determine whether it will halt.

Is this computably decidable?

Given program p and the input, we could run the program, and wait for it to halt.

This would semi-decide the problem: it gives us the Yes answers.

Halting problem

Consider the famous *halting problem*: given a program p and input, determine whether it will halt.

Is this computably decidable?

Given program p and the input, we could run the program, and wait for it to halt.

This would semi-decide the problem: it gives us the Yes answers.

But can we also compute the No answers?

Halting problem is undecidable

The halting problem is undecidable

There is no computational procedure that correctly determines whether a given program will halt.

Halting problem is undecidable

The halting problem is undecidable

There is no computational procedure that correctly determines whether a given program will halt.

A beautiful self-referential argument

Suppose toward contradiction that we could solve the halting problem. Consider this strange algorithm q: On input p, a program, we ask whether program p would halt if given p itself as input; if yes, then we jump into an infinite loop; if no, then we halt immediately.

Let us run q on input q. It will halt if and only if it doesn't halt. Contradiction.

So the halting problem is undecidable.

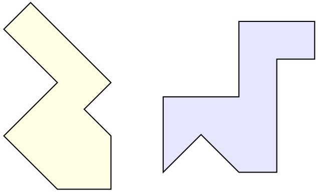
Undecidability

The undecidability phenomenon is now known to be pervasive in mathematics.

We have thousands of concrete decision problems, which cannot in principle be solved by any computational procedure.

Often, one can prove that a problem is undecidable by embedding the halting problem inside it.

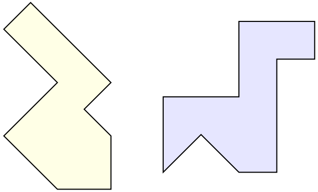
The tiling problem



Tiling problem

Given finitely many polygonal tile types, determine whether they can tile the plane.

The tiling problem

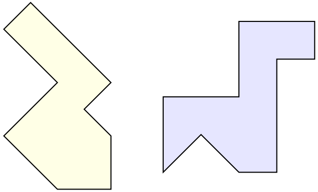


Tiling problem

Given finitely many polygonal tile types, determine whether they can tile the plane.

Question. Is there a computable procedure that will find the answer?

The tiling problem



Tiling problem

Given finitely many polygonal tile types, determine whether they can tile the plane.

Question. Is there a computable procedure that will find the answer?

Answer. No. There is no computational procedure that correctly solves this problem.

More undecidable decision problems

- Halting problem. Decide if a given program halts on a given input.
- Tiling problem. Decide if a given finite set of tile types can tile the plane.

More undecidable decision problems

- Halting problem. Decide if a given program halts on a given input.
- Tiling problem. Decide if a given finite set of tile types can tile the plane.
- Entscheidunsproblem. Decide if a given assertion is provable from a given theory.
- Consistency. Decide if a given theory is consistent.
- Arithmetic Truth. Decide if a given statement of arithmetic is true.

More undecidable decision problems

- Halting problem. Decide if a given program halts on a given input.
- Tiling problem. Decide if a given finite set of tile types can tile the plane.
- Entscheidunsproblem. Decide if a given assertion is provable from a given theory.
- Consistency. Decide if a given theory is consistent.
- Arithmetic Truth. Decide if a given statement of arithmetic is true.
- Diophantine. Decide if a given polynomial equation $p(x_1, \ldots, x_n) = 0$ has a solution in the integers.

More undecidable decision problems

- Halting problem. Decide if a given program halts on a given input.
- Tiling problem. Decide if a given finite set of tile types can tile the plane.
- Entscheidunsproblem. Decide if a given assertion is provable from a given theory.
- Consistency. Decide if a given theory is consistent.
- Arithmetic Truth. Decide if a given statement of arithmetic is true.
- Diophantine. Decide if a given polynomial equation $p(x_1, \ldots, x_n) = 0$ has a solution in the integers.
- Mortality problem. Given a finite set of 3×3 matrices, decide if some product of them (allowing repetitions) is zero.

More undecidable decision problems

- Halting problem. Decide if a given program halts on a given input.
- Tiling problem. Decide if a given finite set of tile types can tile the plane.
- Entscheidunsproblem. Decide if a given assertion is provable from a given theory.
- Consistency. Decide if a given theory is consistent.
- Arithmetic Truth. Decide if a given statement of arithmetic is true.
- Diophantine. Decide if a given polynomial equation $p(x_1, \ldots, x_n) = 0$ has a solution in the integers.
- Mortality problem. Given a finite set of 3×3 matrices, decide if some product of them (allowing repetitions) is zero.
- Elementary expression. Decide if a given functional expression in x using rationals, π, $\ln 2$, $+, -, \times, \sin, \exp,$ abs is identically zero.
- Information. Decide whether a given string can be compressed further.

All these problems are computably undecidable. The proofs generally proceed by reducing the halting problem to each of them.

Turing is credited for undecidability of halting problem

Turing 1936 is widely credited for introducing the halting problem and proving that it is computably undecidable.

Turing is credited for undecidability of halting problem

Turing 1936 is widely credited for introducing the halting problem and proving that it is computably undecidable.

John Stillwell [Sti22, p. 370]

"The problem of deciding whether a given machine halts of a given input—the so-called **halting problem**—must be unsolvable. This result was also observed by Turing (1936)."

Graham Priest [Pri17, pp. 105-107]

"Is there an algorithm we can apply to a program (or, more precisely, its code number) and inputs, to determine whether or not a computation with that program and those inputs terminates? The answer is *no*. And this is what Turing proved. (...) The result is known as the *Halting Theorem*"

Scott Aaronson [Aar13, p. 21], [Aar99]

"Turing's first result is the existence of a "universal" machine (...) But this result is not even the main result of the paper. So what is the main result? It's that there's a basic problem, called the halting problem, that no program can ever solve."

"Turing proved that this problem, called the Halting Problem, is unsolvable by Turing machines. The proof is a beautiful example of self-reference. It formalizes an old argument about why you can never have perfect introspection: because if you could, then you could determine what you were going to do ten seconds from now, and then do something else. Turing imagined that there was a special machine that could solve the Halting Problem. Then he showed how we could have this machine analyze itself, in such a way that it has to halt if it runs forever, and run forever if it halts."

Thomas Cormen [Cor13, p. 210]

"(T)here are problems for which it is provably impossible to create an algorithm that always gives a correct answer. We call such problems ***undecidable***, and the best-known one is the ***halting problem***, proven undecidable by the mathematician Alan Turing in 1937. In the halting problem, the input is a computer program A and the input *x* to A. The goal is to determine whether program A, running on input *x*, ever halts."

Dexter Kozen [Koz12, pp. 243-244]

"The technique of diagonalization was first used by Cantor to show that there are fewer real algebraic numbers than real numbers. Universal Turing Machines and the application of Cantor's diagonalization technique to prove the undecidability of the halting problem appear in Turing's original paper."

Oron Shagrir [Sha06, p. 3]

"[In his 1936 paper] Turing provides a mathematical characterization of his machines, proves that the set of these machines is enumerable, shows that there is a universal (Turing) machine, and describes it in detail. He formulates the halting problem, and proves that it cannot be decided by a Turing machine. On the basis of that proof, Turing arrives, in section 11, at his ultimate goal: proving that the *Entscheidungsproblem* is unsolvable."

Douglas Hofstadter [Hof04, p. XII]

"Fully to fathom even one other human being is far beyond our intellectual capacity — indeed, fully to fathom even one's own self is an idea that quickly leads to absurdities and paradoxes. This fact Alan Turing understood more deeply than nearly anyone ever has, for it constitutes the crux of his work on the halting problem."

Roger Penrose [Pen94, p. 30, our emphasis]

“The mathematical proofs that Hilbert’s tenth problem and the tiling problem are not soluble by computational means are difficult, and I shall certainly not attempt to give the arguments here. The central point of each argument is to show, in effect, how any Turing-machine action can be coded into a Diophantine or tiling problem. *This reduces the issue to one that Turing actually addressed in his original discussion: the computational insolubility of the halting problem*.”

Piergiorgio Odifreddi [Odi92, p. 150]

“The name [of the following theorem] comes from its original formulation, which was in terms of Turing machines, and in that setting it shows that there is no Turing machine that decides whether a universal Turing machine halts or not on given arguments.

Theorem II.2.7 Unsolvability of the Halting Problem (Turing [1936]) *The set defined by* $\langle x, e\rangle \in \mathcal{K}_0 \leftrightarrow x \in \mathcal{W}_e \leftrightarrow \varphi_e(x)\downarrow$ *is r.e. and nonrecursive.*”

Hartley Rogers, Jr

[RJ87, p. 19]. "There is no effective procedure by which we can tell whether or not a given effective computation will eventually come to a stop. (Turing refers to this as the unsolvability of the halting problem for machines. This and the existence of the universal machine are the principal results of Turing's first paper.)"

Hartley Rogers, Jr

[RJ87, p. 19]. “There is no effective procedure by which we can tell whether or not a given effective computation will eventually come to a stop. (Turing refers to this as the unsolvability of the halting problem for machines. This and the existence of the universal machine are the principal results of Turing’s first paper.)”

Joel David Hamkins [Ham21, §6.5]

“Is the halting problem computably decidable? In other words, is there a computable procedure, which on input (p, n) will output yes or no depending on whether program p halts on input n? The answer is no, there is no such computable procedure; the halting problem for Turing machines is not computably decidable. This was proved by Turing with an extremely general argument, a remarkably uniform idea that applies not only to his machine concept, but which applies generally with nearly every sufficiently robust concept of computability.”

Are the attributions accurate?

But did Turing introduce the halting problem and prove it was undecidable?

Are the attributions accurate?

But did Turing introduce the halting problem and prove it was undecidable?

A historical oddity

Almost none of the things attributed to Turing in those quotations are to be found in Turing's paper.

Prima facie case against Turing attribution

- He doesn't define the halting problem or discuss it as a decision problem;
- The phrase "halting problem" does not occur in his paper;
- Indeed, the word "halt" is absent;

Prima facie case against Turing attribution

- He doesn't define the halting problem or discuss it as a decision problem;
- The phrase "halting problem" does not occur in his paper;
- Indeed, the word "halt" is absent;
- He does not discuss halting of machines at all; he makes no provision for his machines ever to stop;
- He has no convention for a *halt* state for his machines;

Prima facie case against Turing attribution

- He doesn't define the halting problem or discuss it as a decision problem;
- The phrase "halting problem" does not occur in his paper;
- Indeed, the word "halt" is absent;
- He does not discuss halting of machines at all; he makes no provision for his machines ever to stop;
- He has no convention for a *halt* state for his machines;
- None of the notation $\varphi_e(x)\downarrow$, $\mathcal{K}_0$, and $\mathcal{W}_e$ occurs in Turing's paper, nor does any equivalent notation appear for these ideas;
- He does not use halting problem to resolve Entscheidungsproblem;

Prima facie case against Turing attribution

- He doesn't define the halting problem or discuss it as a decision problem;
- The phrase "halting problem" does not occur in his paper;
- Indeed, the word "halt" is absent;
- He does not discuss halting of machines at all; he makes no provision for his machines ever to stop;
- He has no convention for a *halt* state for his machines;
- None of the notation $\varphi_e(x)\downarrow$, $\mathcal{K}_0$, and $\mathcal{W}_e$ occurs in Turing's paper, nor does any equivalent notation appear for these ideas;
- He does not use halting problem to resolve Entscheidungsproblem;
- He does not remark on self-contemplative Turing machines;
- Nothing like the self-referential proof of undecidability is found;

Prima facie case against Turing attribution

- He doesn't define the halting problem or discuss it as a decision problem;
- The phrase "halting problem" does not occur in his paper;
- Indeed, the word "halt" is absent;
- He does not discuss halting of machines at all; he makes no provision for his machines ever to stop;
- He has no convention for a *halt* state for his machines;
- None of the notation $\varphi_e(x)\downarrow$, $\mathcal{K}_0$, and $\mathcal{W}_e$ occurs in Turing's paper, nor does any equivalent notation appear for these ideas;
- He does not use halting problem to resolve Entscheidungsproblem;
- He does not remark on self-contemplative Turing machines;
- Nothing like the self-referential proof of undecidability is found;
- All his undecidability arguments instead use the circle-free problem, not even computably equivalent to the halting problem.

The circle-free problem

The central undecidability result of Turing's paper concerns the *circle-free* problem.

The circle-free problem

The central undecidability result of Turing's paper concerns the *circle-free* problem.

A program is circle-free, when it succeeds in giving us infinitely many digits of a computable real number.

Circle-free is undecidable

Turing proves the circle-free problem is undecidable.

Circle-free is undecidable

Turing proves the circle-free problem is undecidable.

He mounts a computable analogue of Cantor diagonalization.

Circle-free is undecidable

Turing proves the circle-free problem is undecidable.

He mounts a computable analogue of Cantor diagonalization.

Namely, if circle-free were decidable, we could computably generate a list of all computable real numbers.

Circle-free is undecidable

Turing proves the circle-free problem is undecidable.

He mounts a computable analogue of Cantor diagonalization.

Namely, if circle-free were decidable, we could computably generate a list of all computable real numbers.

And then we could diagonalize against this list to create a real number that is not on the list.

Circle-free is undecidable

Turing proves the circle-free problem is undecidable.

He mounts a computable analogue of Cantor diagonalization.

Namely, if circle-free were decidable, we could computably generate a list of all computable real numbers.

And then we could diagonalize against this list to create a real number that is not on the list.

Since our process is computable, this is a contradiction! □

Circle-free problem is strictly harder than the halting problem

A curious note: the circle-free problem is actually strictly higher in the hierarchy of computability than the halting problem.

Circle-free problem is strictly harder than the halting problem

A curious note: the circle-free problem is actually strictly higher in the hierarchy of computability than the halting problem.

It is a strictly harder problem.

Circle-free problem is strictly harder than the halting problem

A curious note: the circle-free problem is actually strictly higher in the hierarchy of computability than the halting problem.

It is a strictly harder problem.

So it is a strictly weaker result to show that it is undecidable.

The printing problem

Turing also proves that the *printing problem* is undecidable.

Printing problem

Decide if a given program will ever print a certain symbol during its computation.

The printing problem

Turing also proves that the *printing problem* is undecidable.

Printing problem

Decide if a given program will ever print a certain symbol during its computation.

Turing gives a very clever argument for this.

- It is not a reduction of the circle-free problem to the printing problem (and indeed this is impossible)
- Rather, it is a reductio: if printing problem were decidable, then so to would be the circle-free problem.

Alternative halting criterion

The printing problem is computably equivalent to the halting problem.

Alternative halting criterion

The printing problem is computably equivalent to the halting problem.

In standard formalism, halting is an event that occurs with transition to the *halt* state.

But it could easily have been formalized by printing a certain special *halt* symbol.

Alternative halting criterion

The printing problem is computably equivalent to the halting problem.

In standard formalism, halting is an event that occurs with transition to the *halt* state.

But it could easily have been formalized by printing a certain special *halt* symbol.

So the halting problem is easily seen as essentially equivalent to the printing problem.

We might all have been talking about the printing problem everywhere instead of the halting problem.

Self-referential proof of undecidability for printing

We can mimic the self-referential proof with the printing problem.

Self-referential proof of undecidability for printing

We can mimic the self-referential proof with the printing problem.

The printing problem is computably undecidable

Assume toward contradiction that the printing problem is decidable. Consider algorithm q which on input p, a program, asks whether p on input p would ever print $\downarrow$. If so, then q halts without printing $\downarrow$; but if not, then q prints $\downarrow$ immediately. So q has opposite behavior on input p than p has on input p. So q on input q prints $\downarrow$ if and only if it does not. Contradiction.

Mathematical attribution practice

A cultural observation: mathematicians are often generous in their attributions.

They attribute results, ideas, methods to earlier thinkers, even when those thinkers didn't actually quite do it, but the ideas grow directly out of the earlier work.

Mathematical attribution practice

A cultural observation: mathematicians are often generous in their attributions.

They attribute results, ideas, methods to earlier thinkers, even when those thinkers didn't actually quite do it, but the ideas grow directly out of the earlier work.

Turing didn't consider the halting problem, but he provided all the tools and ideas that we need to prove undecidability ourselves.

Attribution examples

1. Irrationality of $\sqrt{2}$ attributed to the Pythagoreans.
2. Chinese remainder theorem credited to 5th century mathematician Sunzi.

Attribution examples

1. Irrationality of $\sqrt{2}$ attributed to the Pythagoreans.
2. Chinese remainder theorem credited to 5th century mathematician Sunzi.
3. Euler credited with graph theory, 1736 Königsberg bridge problem.
4. Cayley-Hamilton theorem, but Hamilton only treated quaternions, Cayley only state 3×3 case, proved only 2×2 case.

Attribution examples

1. Irrationality of $\sqrt{2}$ attributed to the Pythagoreans.
2. Chinese remainder theorem credited to 5th century mathematician Sunzi.
3. Euler credited with graph theory, 1736 Königsberg bridge problem.
4. Cayley-Hamilton theorem, but Hamilton only treated quaternions, Cayley only state 3×3 case, proved only 2×2 case.
5. Fundamental theorem of finite games attributed to Zermelo 1913.
6. Gödel is credited with improved strong versions of incompleteness theorem.
7. Hilbert spaces.

A nuanced conclusion

Strictly speaking, Turing did not prove nor even state the undecidability of the halting problem in his 1936 paper [Tur36].

A nuanced conclusion

Strictly speaking, Turing did not prove nor even state the undecidability of the halting problem in his 1936 paper [Tur36].

It is incorrect to suggest that this result or any discussion of it can be found there.

Especially incorrect to attribute to Turing [Tur36] the common self-referential proof of undecidability.

A nuanced conclusion

Strictly speaking, Turing did not prove nor even state the undecidability of the halting problem in his 1936 paper [Tur36].

It is incorrect to suggest that this result or any discussion of it can be found there.

Especially incorrect to attribute to Turing [Tur36] the common self-referential proof of undecidability.

Nevertheless, Turing provided a robust framework of ideas sufficient to lead to the undecidability result.

And he proved the undecidability of the printing problem, easily viewed today as computably equivalent to the halting problem.

A nuanced conclusion

Strictly speaking, Turing did not prove nor even state the undecidability of the halting problem in his 1936 paper [Tur36].

It is incorrect to suggest that this result or any discussion of it can be found there.

Especially incorrect to attribute to Turing [Tur36] the common self-referential proof of undecidability.

Nevertheless, Turing provided a robust framework of ideas sufficient to lead to the undecidability result.

And he proved the undecidability of the printing problem, easily viewed today as computably equivalent to the halting problem.

In light of this, we find it correct to say:

> *Turing essentially proved the undecidability of the halting problem in [Tur36].*

Thank you.

Slides and articles available on http://jdh.hamkins.org.

Joel David Hamkins
O'Hara Professor of Logic
University of Notre Dame

References I

[Aar13] Scott Aaronson. *Quantum computing since Democritus*. Cambridge University Press, 2013.

[Aar99] Scott Aaronson. *Who Can Name the Bigger Number?* 1999. https://www.scottaaronson.com/writings/bignumbers.html.

[Cor13] Thomas H Cormen. *Algorithms unlocked*. MIT Press, 2013.

[Ham21] Joel David Hamkins. *Lectures on the Philosophy of Mathematics*. MIT Press, 2021. ISBN: 9780262542234. https://mitpress.mit.edu/books/lectures-philosophy-mathematics.

[HN24] Joel David Hamkins and Theodor Nenu. "Did Turing prove the undecidability of the halting problem?" *Mathematics arXiv* (2024). manuscript under review, 18 pages. arXiv:2407.00680[math.LO]. https://jdh.hamkins.org/turing-halting-problem.

[Hof04] Douglas R. Hofstadter. "Foreword". In: *Alan Turing: Life and legacy of a great thinker*. Ed. by Christof Teuscher. Springer, 2004, pp. IX–XIII.

References II

[Koz12] Dexter C. Kozen. *Automata and computability*. Springer Science & Business Media, 2012.

[Odi92] Piergiorgio Odifreddi. *Classical recursion theory: The theory of functions and sets of natural numbers*. Elsevier, 1992.

[Pen94] Roger Penrose. *Shadows of the Mind*. Vol. 4. Oxford University Press Oxford, 1994.

[Pri17] Graham Priest. *Logic: A very short introduction*. Vol. 29. Oxford University Press, 2017.

[RJ87] Hartley Rogers Jr. *Theory of recursive functions and effective computability*. MIT press, 1987.

[Sha06] Oron Shagrir. "Gödel on Turing on computability". *Church's Thesis after* 70 (2006), pp. 393–419.

[Sti22] John Stillwell. *The story of proof: logic and the history of mathematics*. Princeton University Press, 2022.

References III

[Tur36] A. M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society* 42.3 (1936), pp. 230–265. ISSN: 0024-6115. DOI: 10.1112/plms/s2-42.1.230.