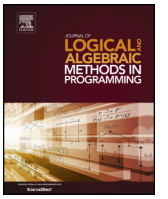
Journal of Logical and Algebraic Methods in Programming 121 (2021) 100687

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp



The origins of the halting problem ☆

Salvador Lucas

DSIC & VRAIN, Universitat Politècnica de València, Valencia, Spain

A R T I C L E I N F O

Article history:
Received 2 November 2020
Received in revised form 1 May 2021
Accepted 2 May 2021
Available online 10 May 2021

Keywords:
Halting problem
Printing problem
Program termination

A B S T R A C T

The *halting problem* is a prominent example of undecidable problem and its formulation and undecidability proof is usually attributed to Turing's 1936 landmark paper. Copeland noticed in 2004, though, that it was so named and, apparently, first stated in a 1958 book by Martin Davis. We provide additional arguments partially supporting this claim as follows: (i) with a focus on computable (real) numbers with infinitely many digits (e.g., π), in his paper Turing was *not concerned* with halting machines; (ii) the two decision problems considered by Turing concern the ability of his machines to produce specific kinds of *outputs*, rather than reaching a halting state, something which was missing from Turing's notion of computation; and (iii) from 1936 to 1958, when considering the literature of the field no paper refers to any "halting problem" of Turing Machines until Davis' book. However, there were important preliminary contributions by (iv) Church, for whom termination was part of his notion of computation (for the λ-calculus), and (v) Kleene, who essentially formulated, in his 1952 book, what we know as the halting problem now.

1. Introduction

In teaching courses about programming languages, program analysis, and software engineering, the halting problem is frequently invoked to justify the

> *well-known piece of folk-lore among programmers that it is impossible to write a program which can examine any other program and tell, in every case, if it will terminate or get into a closed loop when it is run.* [18]

This is often given as a simplified definition of the problem, as Turing machines and computability issues are poorly known by newcomer students or practitioners. In contrast, the practical relevance of such *program termination problem* is easily understood as soon as one points to well-known misconducts of programs like "hanging executions", which often hide neverending loops or infinite recursions, i.e., an undesired termination behavior (see, e.g., [1, page 24] for an informal presentation of the halting problem in this vein). Besides its obvious interest for specialists in program termination, and its use to provide a first understanding of undecidability issues, other authors acknowledge the halting problem as "*one of the most philosophically important theorems of the theory of computation*" [17, Section 4.2].

The formulation and undecidability proof of the halting problem are usually referred to the landmark Turing's paper [19], that introduces his *a-machines*, soon renamed as *Turing Machines* by Church in a 1937 review of [19].[1] However, [4] remarks

☆ Partially supported by the EU (FEDER), and projects RTI2018-094403-B-C32, PROMETEO/2019/098.
E-mail address: slucas@dsic.upv.es.

[1] [4], footnote 3.

https://doi.org/10.1016/j.jlamp.2021.100687

that, although "*it is often said that Turing stated and proved the halting problem in his 1936 paper*", it was named and originally stated by Martin Davis in [7].[2]

Recent books about Turing's works and contributions already acknowledge this fact, see, e.g., [1, page 13]. Some papers about program termination now include some mild warnings about the usual association of the halting problem to Turing, see, e.g., [3, footnote b]. Still, many papers and textbooks refer the definition and proof of undecidability of the halting problem to [19].

We agree with Copeland that crediting the authorship of the definition and proof of unsolvability of the halting problem to Turing is *not* correct. This claim is substantiated on the basis of three different arguments:

1. a *motivational* point of view, where we claim that, from the material in [19], we would rather say that Turing was not especially interested in halting machines; we believe that this is the reason why, as Copeland observes, he pays no attention to this notion in his 1936 paper; then
2. we show that the decidability problems considered by Turing are conceptually different from the halting problem, and, finally,
3. from a *bibliographical* point of view, we notice the absence of references to any "halting problem" in the considered bibliography from [19] to the first publication referring a "halting problem", i.e., [7] (with the remarkable exception of [14] discussed below).

However, we also discuss the following related issues:

4. The idea of a (Turing) machine reaching a stopping state is missing from Turing's definition of computation. In contrast, it is part of Church's definition of *effective calculation* in the λ-calculus [2].
5. In his 1952 book [10] Kleene provides a statement which we easily recognize as what is called "halting problem" today.

In the following, we develop these points.

2. Turing machines and Turing's notion of computability

A Turing machine (see [19, Section 1] but also [13]) is a device that operates on an infinite *tape* arranged as an infinite sequence of possibly empty *cells*. A *head* is able to *scan* the tape by reading and writing (or *printing*) symbols; it moves step by step along the tape. The machine is able to adopt finitely many *states* $q_1, \ldots, q_n$ (*m-configurations* in Turing's terminology). The *operations* performed on the tape (i.e., printing symbols and/or moving to the left or to the right), and also possible changes of state, are determined by the so-called configuration q_i, s_j of the machine when operating in the state q_i and the scanned symbol in the j-th cell of the tape is s_j. The a-machines, usually called (simple) Turing Machines, TM in the following, are those whose motion is completely determined by the configuration.[3]

Formally, the machine can be seen as a finite set of quadruples[4] $\langle q, s, op, q' \rangle$. Here, q and q' are m-configurations (or *states* in the current terminology), s is a symbol from a finite alphabet Σ of symbols $s_1, \ldots, s_k$ (enriched with a special symbol B, often written s_0, denoting a *blank cell* in the tape). Operations op can be L, R, or Ps, where L and R prescribe a one-cell-move of the head to the left or to the right of its current position in the tape, and Ps (or just s) prints symbol s in the scanned cell.

The *Standard Description* (S.D.) of a TM provides a full representation of the (behavior of the) machine [19, page 240]. States q_i are represented by the letter 'D' followed by i occurrences of letter 'A'. Similarly, symbols s_j are represented by the letter 'D' followed by j occurrences of 'C'. In this way, a sequence of quadruples representing a TM becomes a sequence of letters. By replacing each of these letters by appropriate numbers we obtain a *Description Number* (D.N.) of the machine, which uniquely determines its S.D. and structure.

2.1. Computation of sequences

Turing's notion of computation pays no attention to any halting behavior. In the subsection about *computing machines*, i.e., [19, Section 2], given an automatic machine, he writes [19, page 232]:

> *if the machine is supplied with a blank tape and set in motion, starting from the correct initial m-configuration, the subsequence of the symbols printed by it which are of the first kind*[5] *will be called the sequence computed by the machine.*

[2] [4], page 40 and footnote 61. In personal communication to Copeland, Davis thinks likely that he used the term halting problem for the first time during a series of lectures in 1952.

[3] Turing also considered c-machines ('c' standing for choice), where some steps may depend on arbitrary choices made by an external operator [19, Section 2].

[4] Turing's presentation uses *quintuples*, here we follow the simpler Post's description in [13].

[5] Such symbols "of the first kind" are introduced in page 232: they are called *figures* and restricted to be either 0 or 1.

The machine is not required to halt, nor the subsequence of symbols of the "first kind" (the *figures*, i.e., 0, 1,...) is required to be finite. In order to 'see' the result of the computation, we only need to consider the (possibly infinite) subsequence of figures printed by the machine.

Furthermore, some lines below, in the subsection about *circular* and *circle-free* machines, he differentiates these two kinds of machines, and defines the notion of computable sequence as one computed by a circle-free machine.

2.2. Circular and circle-free machines

The notions of *circular* and *circle-free* machines are given in [19, page 233]:

> *if a computing machine never writes down more than a finite number of symbols of the first kind, it will be called circular. Otherwise, it is called circle-free.*

At first sight, the previous sentence may suggest that circular machines always halt. However, the clarification that immediately follows denies this:

> *A machine will be circular if it reaches a configuration from which there is no possible move, or it goes on moving, and possibly printing symbols of the second kind.*[6]

Example 1 *(Circular and circle-free machines).* Examples of circular machines (given as programs)[7] are:

$\mathcal{P}_{C1}$: `skip`

which stops after doing nothing (first case of Turing's definition). Then,

$\mathcal{P}_{C2}$: `while true do print #`

which prints infinitely many symbols of the second kind (fits the second case of Turing's definition). Finally,

$\mathcal{P}_{C3}$: `print 0; while true do skip`

which prints `0` only once and then runs forever without printing anything else is circular as well. An example of circle-free machine is

$\mathcal{P}_{CF}$: `while true do print 0`

which prints infinitely many symbols of the first kind.

Therefore, circular machines may not halt; the important point is not halting but *printing* finitely many figures only. Note that circle-free machines *never* halt; the important point now is printing infinitely many figures. Thus,

Fact 1: Turing partitions his machines into two classes, none of which is required to halt.

2.3. Computations with Turing machines

The fact that Turing's partition of his machines pays no special attention to any halting behavior suggests that, at least in [19], *he was not particularly interested in investigating halting machines*. Furthermore, he makes explicit that only (subsequences of) infinite sequences of figures (computed by circle-free machines) are considered as *computable* [19, page 233]:

> *A sequence is said to be computable if it can be computed by a circle-free machine.*

Circular machines may halt, but, by definition, circle-free machines *never stop*. Turing chooses the last ones to define his notion of computable sequence.

Fact 2: Turing's notion of computation did not rely on (actually *rejected*!) any termination requirement.

3. Martin Davis' description of Turing machines and computations

Since the notion of computed sequence (see above) includes taking any subsequence of figures printed by a machine, computing finite sequences is possible but the exact definition of such a sequence is not intrinsically provided by the machine operation. An "external observer" is required to extract the subsequence. This is in sharp contrast with the standard, algorithmic, idea of a computation by a TM, which requires that the machine halts, thus providing a completely defined finite sequence as the one which is obtained when the machine halts, cf. [7, Definition 1.9]:

[6] Symbols "of the second kind" are also introduced in page 232 just as "the others", i.e., those which are not of the first kind. Turing used two symbols of the second kind, namely 'ə' and 'x', see page 234. In this paper we only use #.

[7] To improve readability, for the informal examples of Turing machines, we use a simple imperative language whose syntax is hopefully well-understood for anybody with some familiarity with (imperative) programming languages. This is a usual practice in the literature, see, e.g., [18,8,9].

By a computation of a Turing machine $\mathcal{M}$ is meant a finite sequence of instantaneous descriptions $\alpha_1, \ldots, \alpha_p$ such that $\alpha_i \rightarrow \alpha_{i+1}$ for $1 \leq i < p$ and such that α_p is terminal with respect to $\mathcal{M}$.

Davis' instantaneous descriptions are similar to Turing's complete configurations, and *transitions* '$\rightarrow$' represent Turing's *moves* [19, last line of page 232]. The keypoint here is that the last instantaneous description α_p must be *terminal*, i.e., no transition is possible from it [7, Definition 1.9]. In other words, "*the machine interprets the absence of an instruction as a stop order*" when reaching α_p [7, footnote 1 in page 7]. This is the usual notion of computation with TMs today.

Remark 1 *(Computations in Church's λ-calculus).* The idea of a computation that halts was already proposed by Church in [2], where he formulates his *thesis* on the basis of the use of the λ-calculus as a suitable notion of effective method. Computations with the λ-calculus (*effective calculations* in Church's terminology) start with an expression, perform some reduction steps, and finish when a *normal form* (i.e., an expression that cannot be reduced) is obtained. Thus, Church's notion of computation assumes that only *finite* sequences of reductions are considered.[8] In an appendix to [19] added in August 28, 1936 (already in Princeton), Turing proves that every "λ-definable sequence is computable" and vice versa.

Remark 2 *(Kleene's description of the operation of a TM).* In [10] Turing's complete configurations are called *situations* by Kleene. Accordingly, TMs are used as follows [10, page 358]:

The change from the initial situation to the terminal situation (when there is one) may be called the operation *performed by the machine.*

Here, the *terminal situation* or *output* is one "*in which [the machine] stops*". Changes between intermediate situations are performed in the usual way, using Turing's *moves*. Davis' notion of "computation of a TM", then, is analogous to Kleene's notion of "operation performed by a TM".

4. Decision problems about Turing machines

In his paper [19], Turing enunciates two problems about his machines (now called *satisfactoriness* and *printing* problems) and proves them undecidable. We discuss them in the next two subsections. The third subsection introduces Davis' definition of the halting problem. The fourth subsection compares them.

4.1. The satisfactoriness problem

The first problem considered by Turing is given in page 247, as follows:

Is there a machine $\mathcal{D}$ which, when supplied with the S.D. of any computing machine $\mathcal{M}$ will test this S.D. and if $\mathcal{M}$ is circular will mark the S.D. with the symbol 'u' and if it is circle-free will mark it with 's'?

Here,'u' and 's' represent an *unsatisfactory* or *satisfactory* verdict of $\mathcal{D}$ about $\mathcal{M}$ being circle-free. Copeland coined this as the *Satisfactoriness Problem* [4, page 36]. In the following, we refer to it as SATIS.

4.2. The printing problem

The second problem, considered in [19, page 248], is what Turing used to show the *Entscheidungsproblem* impossible:

there can be no machine $\mathcal{E}$ which, when supplied with the S.D. of an arbitrary machine $\mathcal{M}$, will determine whether $\mathcal{M}$ ever prints a given symbol (0 say).

Davis uses *printing problem* (PRINT in the following) to refer to it [7, page 70]. In the following, for each machine $\mathcal{M}$, we say that Print($\mathcal{M}$) is *true* iff $\mathcal{M}$ prints a given symbol, (e.g., 0) during its (finite or infinite) execution (and often say that $\mathcal{M}$ is a *printing* machine). For instance, both Print($\mathcal{P}_{C1}$) and Print($\mathcal{P}_{C2}$) are *false*, but Print($\mathcal{P}_{C3}$) and Print($\mathcal{P}_{CF}$) are *true*.

4.3. The halting problem

According to Davis' formulation, the halting problem for a TM $\mathcal{M}$ aims [7, page 70]

to determine whether or not $\mathcal{M}$, if placed in a given initial state, will eventually halt.

[8] Turing wrote [19] in England, before moving to Princeton to pursue a PhD with Church. Thus, Church's and Turing's notions of computation grew independently. We may think of Davis' definition as providing a kind of consensus among them: it is based on halting Turing machines.

This is the standard understanding of the halting problem for TMs (HALT in the following). Davis proves it undecidable in Chapter 5, Theorem 2.2. In the following, we say that $\mathsf{Halt}(\mathcal{M})$ is *true* iff $\mathcal{M}$ halts when placed in a given initial state (and often say that $\mathcal{M}$ is a *halting* machine). For instance, $\mathsf{Halt}(\mathcal{P}_{C1})$ is *true*, but $\mathsf{Halt}(\mathcal{P}_{C2})$, $\mathsf{Halt}(\mathcal{P}_{C3})$, and $\mathsf{Halt}(\mathcal{P}_{CF})$ are *false*.

4.4. *Relationship between* printing *and* halting *problems*

In his book [7], Davis first introduces the *halting* problem and then the *printing* problem. He does not mention whether Turing introduced one or the other. When introducing the printing problem, he calls it "*a related problem*" (to the previously introduced halting problem). The presentation he gave of the printing problem in [7, page 70, last paragraph], i.e.,

> *To determine, of a given instantaneous description of $\mathcal{M}$, whether or not there exists a sequence $\alpha_1, \ldots, \alpha_k$ of instantaneous descriptions such that $\alpha = \alpha_1$, $\alpha_{j-1} \rightarrow \alpha_j$ for $1 < i \leq k$, and α_k contains the symbol s.*

clearly fits Turing's definition as given above (provided that s is 0, although Turing's formulation, "*prints a given symbol (0 say)*", does not suggest a strong dependency of the problem on any specifically chosen symbol). Davis does not further develop the relationship between the halting and printing problems (which he considers 'related'). Although, implicitly, this already means that both problems should *not* be considered identical, in the following, we point to some specific differences between them.

4.4.1. *Printing and halting machines do not coincide*

There are machines $\mathcal{M}$ which print something but do not halt. For instance, the machines associated to programs $\mathcal{P}_{C3}$ and $\mathcal{P}_{CF}$. Therefore,

$$\mathsf{Print}(\mathcal{M}) \not\Rightarrow \mathsf{Halt}(\mathcal{M}) \tag{1}$$

Vice versa, there are machines (e.g., $\mathcal{P}_{C1}$) which stop without printing anything. Hence,

$$\mathsf{Halt}(\mathcal{M}) \not\Rightarrow \mathsf{Print}(\mathcal{M}) \tag{2}$$

Fact 3: Printing and halting machines do *not* coincide.

4.4.2. *Decidability of the halting and printing problems differs*

There are machines $\mathcal{M}$ whose printing problem is decidable, but whose halting problem is *not* decidable. For instance, consider the machine $\mathcal{M} = \{\langle q_0, s, 0, q_0' \rangle\} \cup \mathcal{M}'$, where the initial state of $\mathcal{M}'$ is q_0'. If the halting problem of $\mathcal{M}'$ is undecidable, then the halting problem of $\mathcal{M}$ also is. However, the printing problem for $\mathcal{M}$ is trivially decidable as we only need to notice that, with $\mathcal{M}$ in the initial state q_0, a figure 0 is immediately printed before entering into $\mathcal{M}'$ computations. Therefore,

Fact 4: There are Turing Machines for which the printing problem is decidable, but the halting problem is not.

According to this discussion, it is clear that, from a conceptual point of view, the printing and halting problems address different issues and exhibit important conceptual and technical differences. However, regarding the degree of *unsolvability* of HALT and PRINT, in the following section we show the existence of a correspondence among them. We also consider SATIS to display a full hierarchy of the three problems.

5. The hierarchy of unsolvability for the halting, printing, and satisfactoriness problems

In his PhD thesis, see [20], Turing introduced the idea of comparing different unsolvable problems by means of a variant of his 1936 *a-machines* (called *o-machines*) where undecidable questions could be 'answered' with the help of an *oracle*, the main idea being that questions Q which are undecidable by a TM could be answered by an *extended* version with some additional knowledge provided by the oracle. Such questions would be strictly harder than problems P which are known to be solvable by a TM. Post coined the term *degree of unsolvability* [12, page 289], although the problem had also been investigated by Kleene (see, e.g., [10] and the references therein).

In the hierarchy of unsolvability, $\mathbf{0}$ represents the class of solvable problems (or recursive sets of numbers) and $\mathbf{0}'$ (called the *jump* of class $\mathbf{0}$) represents the class of recursively enumerable, but not recursive, sets of numbers. We have $\mathbf{0} < \mathbf{0}'$ (informally: $\mathbf{0}'$ is strictly harder than $\mathbf{0}$). The degree of undecidability of HALT is $\mathbf{0}'$, written $\mathsf{HALT} \equiv \mathbf{0}'$, see, e.g., [16, page 262].

In the following, we show that HALT, PRINT, and SATIS are related as follows:

$$\mathsf{HALT} \equiv \mathsf{PRINT} \equiv \mathbf{0}' < \mathbf{0}'' \equiv \mathsf{SATIS} \tag{3}$$

where $\mathbf{0}''$ denotes an unsolvability degree strictly bigger (harder) than $\mathbf{0}'$ (see below for an example).

5.1. HALT *and* PRINT *have the same undecidability degree*

Davis proves the printing problem unsolvable as a consequence of the unsolvability of the halting problem [7, Chapter 5, Theorem 2.3].[9] Starting from a halting machine $\mathcal{M}$ he shows how to obtain a machine $\mathcal{M}'$ that prints a given symbol. In this way, he *reduces* the halting problem to the printing problem, written $\mathsf{HALT} \leq \mathsf{PRINT}$. With a similar construction we show $\mathsf{PRINT} \leq \mathsf{HALT}$. Let $q_1, \ldots, q_k$ be the states of $\mathcal{M}$. A machine $\mathcal{M}'$ (with alphabet $\Sigma = \{0, 1\}$) is obtained as follows. Let Γ be the set of 'printing' quadruples of $\mathcal{M}$, as follows (without loss of generality, we focus on printing '0'):

$$\langle q_i, s, 0, q_j \rangle$$

for $s \in \Sigma$ and $i, j = 1, 2, \ldots, k$. That is, Γ consists of the quadruples of $\mathcal{M}$ that print '0'. We obtain $\mathcal{M}'$ from $\mathcal{M}$ in two steps. First, replace in $\mathcal{M}$ each quadruple belonging to Γ by a quadruple

$$\langle q_i, s, 0, q_H \rangle$$

where q_H is a new halting state. Now, let $\mathcal{M}^{\downarrow}$ be the set of quadruples of $\mathcal{M}$ that could lead $\mathcal{M}$ to use a quadruple in Γ, possibly after some moves. We replace each quadruple $\langle q_i, s, s', q_j \rangle$ in $\mathcal{M}$ which is *not* in $\mathcal{M}^{\downarrow}$ by a new quadruple $\langle q_i, s, s', q_L \rangle$, where q_L is a new looping state. Finally, we add to $\mathcal{M}'$ quadruples $\langle q_L, s, s', q_L \rangle$ for all symbols s and s'. Thus, $\mathcal{M}$ eventually prints 0 if and only if $\mathcal{M}'$ halts. We conclude $\mathsf{PRINT} \leq \mathsf{HALT}$. Thus, $\mathsf{HALT} \equiv \mathsf{PRINT}$.

Fact 5: HALT and PRINT have the same undecidability degree.

5.2. *The undecidability degree of* SATIS *is* $\mathbf{0}''$

Turing's proof of undecidability of PRINT [19, page 248] relies on the undecidability of SATIS.

Remark 3 *(Post's conjecture).* As noticed by Post, Turing's method of proof was *reductio ad absurdum* rather than *reduction* as above [13, footnote 14]. Indeed, Post remarks that SATIS is "*almost surely of higher-degree of unsolvability*" than PRINT (written $\mathsf{PRINT} < \mathsf{SATIS}$). Thus, a reduction of SATIS to PRINT would not be possible.

In the following, we prove Post's conjecture. Actually, we prove that $\mathsf{SATIS} \equiv \mathbf{0}''$. As far as we know, this is the first time it is formally proved in the literature. Consider the following problem concerning the characterization of TMs that compute total (monadic) functions [8, pages 378-379]:

$$\mathsf{TOT} = \{z \in \mathbb{N} \mid \Theta(x, z) \text{ is defined for all } x\}$$

Here, $\Theta(x, z)$ is the outcome of the computation of a machine $\mathcal{M}$ with code (D.N.) z and input x [8, page 55]. Davis and Weyuker prove TOT unsolvable. Furthermore, they prove $\mathsf{TOT} \equiv \mathbf{0}''$ [8, page 392].

Remark 4. TOT is equivalent to the *uniform halting problem* (UHALT), i.e., the problem investigating whether a TM halts for *every* input, see, e.g., [11, page 57]. Hence, $\mathsf{UHALT} \equiv \mathsf{TOT}$.

First, we show that TOT is reducible to SATIS (i.e., $\mathsf{TOT} \leq \mathsf{SATIS}$) by means of a machine $\mathcal{S}_{\mathcal{M}}$ which we define in two steps.

1. Let Γ be the set of quadruples $\langle q, s, s', q' \rangle$ which are *not* in $\mathcal{M}$, and $\mathcal{M}^{\downarrow}$ be the set of quadruples of $\mathcal{M}$ leading, after some moves with $\mathcal{M}$, to a (halting) configuration $q\,s$ with $\langle q, s, s', q' \rangle$ in Γ for some s' and q'. Let $\mathcal{M}_1$ be $\mathcal{M}$ after replacing each $\langle q_i, s, s', q_j \rangle \in \mathcal{M} - \mathcal{M}^{\downarrow}$ by $\langle q_i, s, B, q_L \rangle$. Then, $\mathcal{M}'$ is obtained from $\mathcal{M}_1$ by adding quadruples $\langle q_L, s, B, q_L \rangle$ for all possible symbols s and a new 'looping state' q_L. Note that $\mathcal{M}'$ halts if $\mathcal{M}$ does, but if $\mathcal{M}$ does not halt, then $\mathcal{M}'$ ultimately prints infinitely many blanks only.
2. We obtain $\mathcal{S}_{\mathcal{M}}$ by combining $\mathcal{M}'$ and a new machine $\mathcal{N}$ which generates all possible numbers one by one ([19, page 234] defines a similar one); after generating each number, $\mathcal{M}'$ is applied to write an output. If $\mathcal{M}$ is 'total' then the infinite generation of figures implemented by $\mathcal{N}$ makes $\mathcal{S}_{\mathcal{M}}$ circle-free. Otherwise, the generation of figures is stopped by $\mathcal{M}'$ and $\mathcal{S}_{\mathcal{M}}$ is not circle-free.

Thus, $\mathcal{M}$ defines a total function if and only if $\mathcal{S}_{\mathcal{M}}$ is circle-free: $\mathsf{TOT} \leq \mathsf{SATIS}$. This already confirms Post's claim.

Now, we prove $\mathsf{SATIS} \leq \mathsf{TOT}$. Given a TM $\mathcal{M}$ we obtain a machine $\mathcal{T}_{\mathcal{M}}$ in two steps as follows:

[9] More precisely, Davis proves the existence of a machine whose printing problem is undecidable from the existence of a machine whose halting problem is undecidable (which he also shows).

$$\mathsf{HALT} \equiv \mathsf{PRINT} \equiv \mathbf{0}' < \mathbf{0}'' \equiv \mathsf{TOT} \equiv \mathsf{UHALT} \equiv \mathsf{SATIS}$$

Fig. 1. Undecidability hierarchy for PRINT, and SATIS, also with TOT and UHALT.

1. A new machine $\mathcal{M}'$ is obtained by first extending $\mathcal{M}$ with new quadruples $\langle q, s, B, q_L\rangle$ for each halting configuration $q\,s$ of $\mathcal{M}$, where q_L is a new looping state. Now, add $\langle q_L, s, B, q_L\rangle$ for each symbol s to obtain $\mathcal{M}'$. Note that $\mathcal{M}'$ does not halt. If $\mathcal{M}$ does not halt, $\mathcal{M}'$ and $\mathcal{M}$ print the same symbols when starting from any state originally in $\mathcal{M}$. However, if $\mathcal{M}$ halts, then $\mathcal{M}'$ enters into a loop and prints infinitely many blank symbols only.
2. Now, given a natural number n represented on the tape (by a sequence of n digits 0), $\mathcal{T}_{\mathcal{M}}$ executes $\mathcal{M}'$ and halts when exactly n figures have been printed by $\mathcal{M}'$.

If $\mathcal{M}$ is circle-free, then $\mathcal{M}'$ produces infinitely many figures and $\mathcal{T}_{\mathcal{M}}$ is total. If $\mathcal{M}$ is not circle-free, then $\mathcal{M}'$ prints finitely many figures only, but it does not halt. Therefore, there will be a number n_0 for which the inspection of $\mathcal{T}_{\mathcal{M}}$ over the tape of $\mathcal{M}'$ whilst simulating its moves never ends. Thus, $\mathcal{T}_{\mathcal{M}}$ is not total. Hence,

$$\mathsf{SATIS} \equiv \mathbf{0}'' \tag{4}$$

Fact 6: The undecidability degree of SATIS is strictly bigger than the undecidability degree of HALT and PRINT.

The full hierarchy of the considered problems regarding their undecidability degrees is shown in Fig. 1.

6. Bibliographical analysis

The (necessarily incomplete) analysis of the literature about computability and undecidability for the period from 1936 to 1958 shows that, apparently, the term "halting problem" was not used before the publication of [7]. A number of journals and repositories in the fields of mathematics, logic, and computer science with facilities to search for (prefixes of) words in their archives were considered and '*halt*' tried on the corresponding search devices for the period 1936-1958. In particular, journals which were available in this period like *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, *Acta Mathematica*, *Acta Mathematica Hungarica*, *Acta Scientiarum Mathematicarum*, *American Journal of Mathematics*, *American Mathematical Society*, *Archive For Mathematical Logic*, *The Journal of Symbolic Logic*, *Mathematics of Computation* were considered for this purpose. The obtained outcomes had nothing to do with the halting problem.

Fact 7: Apparently, the term "halting problem" was not used in the literature before the publication of [7] (but see the discussion below).

Remark 5 *(Kleene's statement).* In his 1952 book, Kleene makes the following statement [10, Chapter XIII, Section 71][10]

> there is no algorithm for deciding whether any given machine, when started from any given situation, eventually stops.

which corresponds to what we currently know as the *halting problem*, although the word "*stops*" is used instead of "*halts*". Kleene justifies this statement by using Turing's arguments, but without citing any of his results. This suggests that Kleene does *not* identify any of Turing's results in [19] as being equivalent to the statement above.

Early references by Davis to a preliminary ("in preparation") version of [7] can be found in [5,6]. However, they did not mention the halting problem yet. Remarkably, Rogers' 1957 book [14] (a preliminary version of [16]) includes what apparently is the *first printed reference* to the halting problem. He writes [14, page 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[11] *for machines. This and the existence of the universal machine are the principal results of Turing's first paper.)*

Clearly, Rogers does not claim the authorship of the notion, as he refers to Turing instead. But Copeland already mentioned that Turing never used the word "halting" in [19]. In the preface, Rogers says that the manuscript of [7] was available to him when preparing the book [14, page 3]. Moreover, one year later, in [15, page 333], with [7] already published, he mentions the halting problem again but he does not mention Turing anymore; instead he cites [7].

[10] This pointer was provided by one of the referees.
[11] Underlined in the original.

7. Conclusions

The subject of Turing's paper was "*ostensibly the computable numbers*" [19, page 230]. He defines a number as computable if "*its decimal part can be written down by a machine*"; and points to numbers like π (with infinitely many decimals without any repetition pattern), as computable [19, page 230]. Thus, if Turing's focus was on infinitary computations, it is not surprising that investigating halting machines was not a priority. Furthermore, Turing needed nonterminating machines but only 'productive' ones, in the sense that infinitely many figures are printed during the computation. He defines a number as "*computable if it differs by an integer from the number computed by a circle-free machine*".[12] Accordingly, this is the focus of his first undecidability result: SATIS tries to determine whether a given machine is circle-free (satisfactory!), and hence able to generate a sequence that corresponds to a computable number. Halting machines were just a subclass of *unsatisfactory* machines.

So, why the halting problem should concern him?

Another indirect reason to think like that is Strachey's account of his encounter with Turing in 1953. Although Strachey does not mention the halting problem of TMs as he rather speaks of programs and program termination, as quoted in the introduction, in [18] Strachey writes

> *I have never actually seen a proof of this in print, and though Alan Turing once gave me a verbal proof (in a railway carriage on the way to a Conference at the NPL in 1953), I unfortunately and promptly forgot the details.*

This suggests that Turing's (obviously *printed*) results in [19] were not considered (either by Turing or Strachey, or Kleene, as remarked above) as providing a proof of undecidability of the halting problem.

Summarizing: Church included termination as part of his notion of effective calculation [2], but this concerns λ-calculus as computational mechanism, not Turing machines. He proved termination of effective calculations undecidable.

When introducing his a-machines [19], Turing was not interested in halting machines and his notion of computation focused instead on the generation of infinite sequences of figures. His undecidability results were according to this.

Kleene's notion of computability considered *stopping* as an ingredient of the notion of computation (of a partial number-theoretic function), which could be required or not, thus leading to alternative notions [10, Section 67, in particular pages 361-362]. Still, as discussed in Remark 2, his notion of *operation* of a TM essentially coincides with Davis' notion of computation. He formulated and informally proved a statement (see Remark 5) which we easily recognize as what we call halting problem now (but 'stops' is used instead of 'halts').

Davis' notion of computation required a halting behavior of the machines. It is not surprising, then, that Davis formulated the halting problem and proved it undecidable. Strikingly, the last sentence of [7, Chapter 5, Section 2]:

> *It might also be mentioned that the unsolvability of essentially these problems was first obtained by Turing*

probably contributed to give credit on the halting problem to Turing. However, as discussed in the previous sections, and summarized through *Facts 1–7*, the conceptual differences between Turing's decision problems and the halting problem suggest that this claim just honors Turing's pioneering work in posing and solving relevant problems in Computer Science, including the undecidability of the satisfactoriness and printing problems. However, the halting problem is *not* among the contributions of Turing in [19].

Conclusion: Martin Davis deserves credit for the wording of the current formulation of the halting problem (and also for the somehow standard expression "halting problem"). It seems, however, that it was Stephen C. Kleene the one who first mentioned the problem, although using a different wording (see Remark 5). Alonzo Church made an important, early contribution by requiring termination as part of his notion of computation ("effective calculation") in the λ-calculus (Remark 1).

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

I thank the anonymous referees for their insightful and precise comments, leading to several improvements of the paper; I also thank Alberto Lluch Lafuente for his help and patience. Thanks are also due to María Alpuente, Santiago Escobar, Pascual Julián, and Josep Silva for commenting on earlier versions of the paper. Finally, I thank my wife, Sandra, for her encouragement and support, and also for her comments.

[12] The, perhaps, most obvious examples of "computable numbers" (in our current view), namely the natural numbers, are assumed to be given at once by means of a sequence of symbols which should actually be used as a single symbol [19, page 249] without any real need of computation.

References

[1] C. Bernhardt, Turing's Vision. The Birth of Computer Science, The MIT Press, 2016.
[2] A. Church, An unsolvable problem of elementary number theory, Am. J. Math. 58 (1936) 345–363.
[3] B. Cook, A. Podelski, A. Rybalchenko, Proving program termination, Commun. ACM 54 (2011) 88–98, https://doi.org/10.1145/1941487.1941509.
[4] J.B. Copeland, The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence and Artificial Life: Plus the Secrets of Enigma, Oxford University Press, 2004.
[5] M.D. Davis, A Note on Universal Turing Machines, Princeton University Press, 1956, pp. 167–175.
[6] M.D. Davis, The definition of universal Turing machine, Proc. Am. Math. Soc. 8 (1957) 1125–1126.
[7] M.D. Davis, Computability and Unsolvability, McGraw-Hill, 1958.
[8] M.D. Davis, E.J. Weyuker, Computability, Complexity, and Languages. Fundamentals of Theoretical Computer Science, Academic Press, 1983.
[9] N.D. Jones, Computability and Complexity. From a Programming Perspective, The MIT Press, 1997.
[10] S.C. Kleene, Introduction to Metamathematics, Wolters-Noordof and North-Holland, 1952.
[11] Z. Manna, Mathematical Theory of Computation, McGraw-Hill, 1974, reprinted by Dover, 2003.
[12] E. Post, Recursively enumerable sets of positive integers and their decision problems, Bull. Am. Math. Soc. 50 (1944) 284–316.
[13] E. Post, Recursive unsolvability of a problem of Thue, J. Symb. Log. 12 (1947) 1–11.
[14] H. Rogers, Theory of Recursive Functions and Effective Computability. Vol. 1. Mimeographed. Technology Store, 1957.
[15] H. Rogers, Gödel numberings of partial recursive functions, J. Symb. Log. 33 (1958) 331–341.
[16] H. Rogers, Theory of Recursive Functions and Effective Computability, The MIT Press, 1967.
[17] M. Sipser, Introduction to the Theory of Computation, second edition, Thomson, 2006.
[18] C. Strachey, An impossible program, Comput. J. 7 (1965) 313.
[19] A.M. Turing, On computable numbers, with an application to the Entscheidungsproblem, Proc. Lond. Math. Soc. 42 (1936) 230–265.
[20] A.M. Turing, Systems of logic based on ordinals, Proc. Lond. Math. Soc. 45 (1939) 161–228.