Halting problem*s*: a historical reading of a paradigmatic computer science problem

Liesbeth De Mol[1] and Edgar G. Daylight[2]

[1] CNRS, UMR 8163 Savoirs, Textes, Langage, Université de Lille, liesbeth.demol@univ-lille3.fr

[2] Siegen University, egdaylight@yahoo.com

Introduction (1)

"The **improbable symbolism** of Peano, Russel, and Whitehead, the analysis of proofs by flowcharts spearheaded by Gentzen, the definition of computability by Church and Turing, all inventions motivated by the purest of mathematics, **mark the beginning of the computer revolution**. Once more, we find a confirmation of the sentence Leonardo jotted despondently on one of those rambling sheets where he confided his innermost thoughts: '**Theory is the captain, and application the soldier**.' " (Metropolis, Howlett and Rota, 1980)

Introduction (2)

Why is this 'improbable' symbolism considered relevant in computing?

⇒ Different non-excluding answers...

1. (the socio-historical answers) studying social and institutional developments in computing to understand why logic, or, theory, was/is considered to be the captain (or not) e.g. need for logic framed in CS's struggle for disciplinary identity and independence (cf (Tedre 2015))
2. (the philosophico-historical answers) studying history of computing on a more technical level to understand why and how logic (or theory) are introduced in the computing practices – question: is there something to computing which makes logic epistemologically relevant to it?

⇒ significance of combining the different answers (and the respective approaches they result in)

⇒ In this talk: focus on paradigmatic "problem" of (theoretical) computer science – the halting problem

Introducing the halting problem

Introducing the halting problem

((Usually (but incorrectly - cf this talk) attributed to Turing))

Popular (generic) version of the halting problem:

"*[I]t is impossible to devise a* **uniform** *procedure, or computer program, which can look at any computer program and* **decide** *whether or not that program will ever* **terminate**." (Minsky 1967, p. 153)

(Historically) instance of a recursively unsolvable decision problems:

"*What is required is a clearly stated (type of) problem which can in some special cases be settled by calculation, but for which a* **uniform general computational method** *of solution seems* **unlikely**" (Gandy 1988, p. 60)

⇒ Why is this a fundamental problem in:

- Foundations of mathematics
- Foundations of Computer Science

... and what are their historical and epistemological connections?

Context I: foundations of mathematics

Decision problems in the context of foundations

- **Development of Mathematical Logic** "professional philosophers have taken very little interest in it, presumably because they found it too mathematical. On the other hand, most mathematicians have taken very little interest in it, because they found it too philosophical" (Skolem, 1928)

- **Decision problems and their impact on math**

  Optimism – Hilbert's dream(s) "The true reason why Comte could not find an unsolvable problem, lies in my opinion in the assertion that there exists no unsolvable problem. Instead of the stupid Ignorabimus, our solution should be: We must know. We will know"' (**Hilbert**, 1930)

  Pessimism – the end of "creative" math? "[T]he contemporary practice of mathematics, using as it does heuristic methods, only makes sense because of this undecidability. When the undecidability fails then mathematics, as we now understand it, will cease to exist; in its place there will be a mechanical prescription for deciding whether a given sentence is provable or not" (**Von Neumann**, 1927)

⇒ Hilbert's dreams failed due to Gödel but also **Post**, **Church**, Kleene and **Turing** – proofs of recursively unsolvable problems

Post's decision problems

1920 - 21: in search for an (algorithmic) solution to the Entscheidungsproblem – formalization of finite symbolic reasoning

- method of simplifying through abstraction ultimately results in Post's **normal form**:

$g_i P_i$	$1101 P_i$	~~1101~~11011101000000
	produces	
$P_i g_{i'}$	$P_i 001$	11011101000000001

- Post's thesis I (Davis 1982) For every set of sequences for which a process can be set-up to generate it, there is also a normal system which will generate it.
- (theorem) "There exists no finite-normal-test for the complete normal system K" – proof by construction of a uniform procedure and diagonalization
- No explicit inclusion of a halting operation

Post's decision problems

1921 (?)-1936: in search of "**complete analysis** *[...] of all the possible ways* **a human mind** *could set up* **finite processes** *for generating sequences*": Formulation 1

- "We have in mind a *general problem* consisting of a class of *specific problems*. A solution of the general problem will thn be one which furnishes an answer to each specific problem. In the following formulation of such a solution..."
- $\sim$ Turing machines but with inclusion of a STOP operation – called when the solution is found. Termination for every instance of the problem is considered to be a necessary condition to identify a given process as a fnite-1-process.
- focus on: finite combinatory processes, not computation or calculation; no machine, computation or calculation talk + no decision problem in (Post 1936);
- "a fundamental discovery in the limitations of the mathematicizing power of Homo Sapiens has been made" (Post 1936)

$\Rightarrow$ (Post 1947) – negative solution to the word problem for semi-groups; phrased in terms of Turing machines which have explicit STOP order (necessary for the proof to work) + "*a number of comparisons with [Post 1936] will occur to a reader of that note*"

Church's decision problems

1932-1933: In search of a purely formal device which avoids Gödel incompleteness:

- "A set of postulates for the foundations of Logic"
- "*[...] This is conceivable on account of the entirely formal character of the system which makes it possible to abstract from the meaning of the symbols and* **to regard the proving of theorems (of formal logic) as a game played with marks on paper according to a certain arbitrary set of rules**" (Church 1933)
- The set turned out to be inconsistent (Kleene and Rosser 1935) – focus on λ-calculus

1936 defining effective calculability

- **Church's thesis** "We now define the notion [...] of an *effective calculable* function of positive integers by identifying it with the notion of a recursive function (or of a λ-definable function of positive integers)"
- **(base) Decision problem** There is no recursive function of a formula $\mathbf{C}$, whose value is 2 or 1 according as $\mathbf{C}$ has a normal form or not
- Relevance? "*symbolic logic in general can be regarded, mathematically, as a branch of elementary number theory*" (Church 1936, fn 8)
- clear refs. to calculability and algorithm talk; no explicit halting order

- used to prove the unsolvability of the Entscheidungsproblem

Turing's decision problems

"*[I]t was Turing alone who [...] gave the first convincing formal definition of a computable function*" (Soare, 2007) – common viewpoint

Turing 1936 (not much prehistory) – "*What are the possible (human) processes that can be carried out in computing a number?*" – formal analysis

- Derivation of a formal notion of an automatic machine (today known as a Turing machine) focusing on five basic human conditions (cfr Sieg and Gandy) to capture notion of computable (real) numbers...
- **Turing's thesis** "*According to my definition, a number is computable if its decimal expansion can be written down by a[n a-]machine*"
- **(base) decision problem** "it is shown that there can be no general process for determining whether a given number is satisfactory or not."
- *a*-machine definition unstable throughout paper – different definitions + so-called "spurious" (Post 1947) Turing convention
- No halt or stop order, in fact **no halting problem**
- BUT (1): proof unsolvability Entscheidungsproblem
- BUT (2) much closer in terminology and spirit to actual computers, including the unversality idea

Thus....

- In its poper context, the halting problem and its associated formalism (thé Turing machine) hides a rich history of different kinds of formalisms, motivations and results. There is no such thing as thé halting problem in this context.
- where is thé "halting problem"?

Thé halting problem

Davis 1958, *Computability and Unsolvability* – influential book in computing

- "the notion of the Turing machine has been made central [...] because of the intuitive suggestiveness of Turing machines and the **analogies** between them and actual digital computers"
- "The present formulation [of Turing machines] (including the use of a two-way infinite tape) follows [(Post 1947)]" (Davis 1958, p. 5, fn 1) – this is in fact Post's formulation I!
- Thé halting problem: "*To determine, of a given instantenuous description α, whether or not there exists a computation of Z that begins with α. That is, we wish to determine whether or not Z [...] will eventually halt. We call the problem the halting problem.*"
- "Another result (the unsolvability of the halting problem) may be interpreted as implying the **impossibility of constructing a program** for determining whether or not an arbitrary given program is free of "loops"."

⇒ **appeal of the "machine" 'metaphor'** which then became not *a* but *thé* model in computing

⇒ Thé halting problem was from its first occurence affiliated with actual computers

Context II: foundations of computer science

Origins of the "first computers" (briefly)

- Number-crunching in "applied" mathematics (e.g. firing tables; navigation; etc): **laborious, lengthy and error-prone** processes of computation (cf e.g. Grier 2007)
- Steady process of mechanization and, ultimately, automatic machines
- e.g. US military in need of firing tables:
  - Computation of firing tables at BRL and Penn: human computers and two differential analyzers (Grier 2007; Polachek 1997)
  - Human work **too slow**, differential analyzer **too inaccurate** (other type of slowness)

⇒ **Pragmatism:** Development of techniques and technologies that fit the purpose: computations that are (partially) **precise (digital), automated (conditional branching and sequencing), fast (electronics)** – the three basic properties of the first computers.

⇒ quite (!) unconnected from developments in mathematical logic

Introducing mathematical logic techniques in computing (in a nutshell) (a)

Around 1946-1948 "first" high-speed "computers"

- Introduction of "alleged" stored-program concept (cf Haigh, Priestley, Rope 2016) – (what is "logical" about it???)
- realization of need for logical approach to programming

⇒ rooted in the high-speed. Cf e.g. von Neumann: *It is evident that in order to be efficient, you will have to treat the problem of logical instructions on the same level on which you treat other memory problems.* (von Neumann 1946)

⇒ Need to "control" possible behaviors through formalism *I think the thing that we learned with this high speed was that ...you had to have a way to program ahead of time [...] a fast computer is useless unless you have some way to program it.* (Alt 1972)

Introducing mathematical logic techniques in computing (in a nutshell) (b)

⇒ **First generation** of logicians/mathematicians turned computer pioneers:

- Haskell B. Curry (see e.g. De Mol, Carlé and Bullynck, 2015)
- John von Neumann (see e.g. Priestley 2018)
- Alan M. Turing (see e.g. Jones 2017)

⇒ In each: elementary forms of "correctness" reasoning, viz. the use of (formal) arguments to reason about the correctness of a program – e.g. assertion boxes (VN) or reasoning about type invariance (Curry)

⇒ **Second generation (1950s)**: several formal models are picked up (+ new ones developed), e.g.:

- λ-calculus in context of programming languages (e.g. McCarthy 1960; Landin 1965)
- Post production systems in the context of formal languages and compiler theory (e.g. Chomsky 1956) and programming languages (Backus 1980)
- Turing machines as a model for computing machinery to reason about minimal instruction sets and machine-independence (e.g. Moore 1952; Carr 1959)

⇒ **Where is the halting problem in all this??**

Termination problems in the computing context

Speedy computation and problem of being "ahead" of the machine → controlling when the machine should STOP and if it will STOP

non-formal and ad hoc approaches:

- an ENIAC example; "It is necessary [...] to incorporate some protection against divergence or excessively slow convergence [...]. This may arise through some error in planning, because we reach the boundary of an interval [...] through some mal-functioning of the Eniac, trhough some roughness in the data, or what not. In such a case it is desirable to so program Eniac as to sense the situation and make necessary adjustments or give a signal"
- development of semi-heuristic methods for doing mathematics due to unpredictability (including termination): "*The limit of 55 stories had to be discovered by the machine itself as well as the "world constant" 23532*" (Lehmer 1963)

"formal"/theoretical approaches?:

- Turing 1949 (cf Jones 2017), 'Checking a large Routine'; '*total correctness*' – "the checker has to **verify that process comes to an end**. [...] [H]e should be assisted by the programmer giving a further definite assertion to be verified. This may take the form of a quantity which is asserted to decrease continually and vanish when the machine stops."

Termination problems in the computing context

⇒ No systematic treatment of termination problems; no refs to thé halting problem

⇒ Need to investigate more closely the 1950s –

⇒ **1960s** termination and correctness problems treated more theoretically

From Programming and modeling machines to Computer Science

1. 1950s (recap)
   (a) computer science a long way from establishing an academically respectable field
   (b) steady evolution of connecting work from 1920s and 1930s to computing machinery – cfr e.g. Automatata studies; abstract machine models; formal language theory; etc. *Some pioneers:* Kleene, Chomsky, Moore, van der Poel
   (c) first PLs (on paper): broader realization need for more formal/linguistic reflections on programming. *Some pioneers:* Saul Gorn, Alan Perlis, John Carr (ps: Gorn and Perlis were both at BRL, Aberdeen proving ground in 1951 when ENIAC was there + Gorn knew Curry)
2. 1960s
   (a) formalizing abstractions away from the machine
   (b) role of maths in engineering: queen rather than servant
   (c) Program Correctness, Language Semantics
      i. John McCarthy, Robert W. Floyd
      ii. Christopher Strachey, C.A.R. Hoare
      iii. Peter Naur, Edsger W. Dijkstra
3. Focus on 1960s and the Halting Problem

Program Correctness: John McCarthy, 1961

1. numerical analysis
2. finite automata
3. theory of computation
   - integer calculation by control flow
     - Church-Kleene formulation
     - Turing machines (Davis'58)
   - symbolic computation
     - McCarthy's recursive function formalism

John McCarthy, 1961

1. "Proving that a recursively defined function converges"
2. "... Gödel's theorem ... "
3. "... proving the equivalence of algorithms ..."
4. "there is no effective general way of deciding whether a process will terminate"

Robert W. Floyd, mid-1960s

1. Perlis, Gorn, McCarthy ⇒ Floyd
2. "a mechanical proof procedure, designed to recognize the elements of a recursively enumerable but not recursive set, cannot be guaranteed to terminate without a fundamental loss of power."

Christopher Strachey, 1965

1. (Turing ⇒ Strachey)
2. "A well-known piece of folklore among programmers ..."
3. "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."

Christopher Strachey, 1965

1. "Suppose T[R] is a Boolean function taking a routine ... as its argument ..."
2. "... this contradiction shows that the function T cannot exist."

Reactions to Strachey, 1965

1. "[Strachey's] letter was of particular interest to me because I had ... proved that it is indeed possible to write such a program ..." — Maurer
2. "Writers of these in the world of practical applications should not let Strachey's formidable piece of generality frighten them off!" — King

C.A.R. Hoare, 1969

1. McCarthy, Floyd ⇒ Hoare
2. With hindsight, Hoare's reception stands out:
   (a) He intertwined computability theory and actual program execution on a finite machine. (Same for Strachey, albeit on an infinite machine.)
   (b) "it is probably better to prove the 'conditional' correctness of a program and rely on an implementation to give a warning if it has had to abandon execution of the program as a result of violation of an implementation limit."
   (c) As we know today: in the general case, the termination problem cannot be solved at runtime.

Program Correctness, Continental Europe, < 1967

1. Peter Naur
2. Edsger Dijkstra
3. Hardly any reception of:
   (a) Turing'36
   (b) Kleene'52
   (c) Davis'58

Discussion

- Halting problems? An intellectual pleasantry or serious business? Significance of proper context!
- Thé halting problem does not exist – shaped and reshaped by history; from a purely theoretical problem to a practical problem that is turned theoretical (cf Hoare and Strachey)

$\Rightarrow$ History opens up the black box of what seem to be "stable" and clear concepts and problems and so opens up our perspectives –

$\Rightarrow$ fixation on one "ultimate" model (TM or VN or ...) blinds us for the need of historicized foundations