Beyond Turing Machines[*]

Kurt Ammon

www.cstruct.org

Abstract

This paper discusses "computational" systems capable of "computing" functions not computable by predefined Turing machines if the systems are not isolated from their environment. Roughly speaking, these systems can change their finite descriptions by interacting with their environment.

1   Introduction

Turing [8] introduced the concept of "computing machines" which subsequently were called Turing machines. He proved that Hilbert's decision problem (Entscheidungsproblem) is unsolvable, that is, there is no Turing machine determining whether or not a given statement in first-order predicate calculus (a mathematical proposition in number theory) can be proved. Wegner [11] writes that Turing's precise characterization of what can be computed established the respectability of computer science as a discipline. He argues that Turing machines cannot capture the intuitive notion of what computers compute when computing is extended to include interaction. His interaction machines have been criticized as an unnecessary Kuhnian paradigm shift [12]. Prasse and Rittgen [7] write that Wegner's "interaction machines cannot compute non-recursive functions, so Church's thesis still holds". This implies that interaction machines cannot "compute" functions not computable by Turing machines.

---

[*]This work is licensed under the Creative Commons Attribution-No Derivative Works 3.0 Unported License (see http://creativecommons.org/licenses/by-nd/3.0/).

This paper proves that there is no Turing machine producing a sequence of all computable functions on the set of natural numbers. The proof implies the existence of "computational" systems that cannot be modeled by predefined Turing machines if the systems are not isolated from their environment. Roughly speaking, these systems change their finite descriptions by interacting with an environment whose development is not completely predictable for practical and theoretical reasons. I argue that the proof even applies to existing systems such as the Internet. Finally, I introduce a new type of systems by requiring that they be capable of "computing" functions not computable by Turing machines.

2 Incompleteness Theorem

A *computable function* is a function that can be computed by a Turing machine, that is, it can be represented by an ordinary computer program. Thus, a computable function on the set of natural numbers $\mathbf{N}$ into $\mathbf{N}$ can be regarded as a computer program producing a natural number in its output from any natural number in its input. An example of such a function f is $f(n) = n + 1$ which produces the successor $n + 1$ of any natural number n.

The computable functions on the set of natural numbers can be regarded as models of computer programs and systems. The restriction to computable functions on the set of natural numbers is not relevant because inputs and outputs of computer programs and systems are represented as binary digits. Useful programs and systems should work for defined sets of inputs which correspond to subsets of the set of natural numbers. Such a definition can be extended to all natural numbers by assuming a default output for inputs for which the program or system is not defined. Ordinarily, a program or system is defined on a decidable set of inputs, that is, there is a program deciding whether or not an input is admissible. Thus, the computable functions on the set of natural numbers model an important class of computer programs and systems.

Incompleteness Theorem: There is no Turing machine producing a sequence of all computable functions f_1, f_2, ... on the set of natural numbers $\mathbf{N}$ into $\mathbf{N}$.

Proof. We assume that there is such a Turing machine. We define a new computable function g on the set of natural numbers by $g(n) = f_n(n) + 1$ for all natural numbers n. Because the sequence f_1, f_2, ... contains all

computable functions according to our original assumption, there is a natural number n such that $g(x) = f_n(x)$ for all natural numbers x. This immediately yields the contradiction $g(n) = f_n(n)$ and $g(n) = f_n(n) + 1$ because of the definition of the function g. This means that our original assumption is false, that is, there is no Turing machine producing all computable functions f_1, f_2, ... on the set of natural numbers.

The construction of the computable function g in the proof can be represented by a Turing machine T. Of course, T can be incorporated into any Turing machine. The proof implies that no Turing machine can produce all computable functions f_1, f_2, ... on the set of natural numbers no matter how T is incorporated. In particular, T can be incorporated into the Turing machine whose existence is assumed in the proof so that the extended Turing machine produces the sequence f_1, f_2, ... and the function g. But the extended machine is different from the original machine and thus its application would contradict our assumption that there is a (single) Turing machine producing all computable functions on the set of natural numbers.

Because any formal system can simply be defined as a theorem-proving Turing machine (see, for example, [4, p. 72]), the theorem also implies that any formal theory is incomplete in the sense that it cannot "capture" all computable functions on the set of natural numbers.

3 Creative Systems

Let C be a system capable of performing the reasoning processes required for proving my simple incompleteness theorem. Thus, C is capable of constructing a computable function g on the set of natural numbers not produced by any given Turing machine M. Figure 1 illustrates the corresponding proof which shows that there is no predefined Turing machine M producing all computable functions f_1, f_2, ... on the set of natural numbers C can construct.

A difference between C and a Turing machine is that C is not regarded as a static predefined object isolated from its environment. Rather, C and the Turing machine M in the proof are regarded as separate interacting entities in the sense that C assumes the existence of a Turing machine M producing all computable functions f_1, f_2, ... on the set of natural numbers in order to construct a computable function g not produced by M. Roughly speaking, C can change itself by interacting with its environment, that is, the Turing

$$C \xrightarrow{\;uses\;} M \xrightarrow{\;produces\;} f_1,\, f_2,\, ...$$

$$\Big\downarrow \; constructs$$

$$g \neq f_1,\, f_2,\, ...$$

Figure 1: Proof

machine M.

Because the capabilities of C to construct computable functions cannot be modeled by any predefined Turing machine, another model of such systems seems to be required.

Computable functions can be represented by Turing machines or computer programs, that is, they have finite descriptions. The set of such descriptions can be effectively enumerated. This means that there is a Turing machine generating a sequence containing all finite descriptions, for example, in ascending length. For these reasons, my simple incompleteness theorem implies that the function deciding whether or not a given description represents a computable function on the set of natural numbers is not computable. If this function were computable, its application to an effective enumeration of descriptions would yield an effective enumeration of all computable functions on the set of natural numbers. This contradicts my theorem. Thus, there is no Turing machine deciding whether or not a given description is a computable function, that is, this decision problem (Entscheidungsproblem) is unsolvable.

Furthermore, my simple incompleteness theorem implies that systems capable of proving this theorem seem to be capable of "solving" the above decision problem *in the sense* that they can construct more and more powerful computable functions on the set of natural numbers beyond the limits of any predefined Turing machine. This suggests the following definition of a new type of systems:

> **Definition:** A system is called *creative* if it is capable of "computing" non-computable functions, that is, determining values of non-computable functions for given arguments.

4

With regard to this definition, creative systems can contain any computable function. Thus, they may be regarded as an extension of the concept of Turing machines in the sense that they can "compute" computable and non-computable functions.

An architecture of creative systems was developed on the basis of experiments with the SHUNYATA program [1, 2]. It is the first step towards the implementation of a creative system. Roughly speaking, a creative system comprises a reflection base and a varying number of evolving analytical spaces.

> **Definition:** The *reflection base* contains a universal programming language, elementary domain-specific concepts, and knowledge about this language and these concepts.

Because all computable functions can be represented in a universal programming language, my simple incompleteness theorem implies that the reflection base cannot be formalized completely.

> **Definition:** *Analytical spaces* contain partial knowledge whose domains of application are limited but ordinarily expand in the course of the development of the analytical spaces.

Roughly speaking, the development of new knowledge in creative systems can be summarized in the following principles:

> **Principles of Development:**
>
> 1. *The knowledge in analytical spaces arises from the reflection base and preceding knowledge.*
>
> 2. *The development of knowledge involves the generation of new analytical spaces and the unification of existing ones.*
>
> 3. *The economical variations of new knowledge tend to be preserved and the uneconomical ones to be destroyed.*

For example, the reflection base may contain elementary knowledge about constants and functions of a programming language.

A very simple programming task is the construction of a program producing the successor $n+1$ of any natural number n in its input. This program can be constructed on the basis of the elementary knowledge that 1 is a natural number and $x + y$ is a natural number for any natural numbers x and y.

Another simple example is the construction of a Quicksort program $sort(L)$, which sorts the elements of a list L according to a given order relation "$x < y$" (x is less than y) between any elements x and y of L. The core of such a program can be represented by the functional pseudocode

$$append(sort(x \in L : x < first(L)),$$
$$first(L),$$
$$sort(x \in L : first(L) < x))), \qquad (1)$$

where the *append* function appends lists and $first(L)$ is the first element of a list L. In order to sort a list L, the program (1) sorts the elements $x \in L$ that are less than its first element $first(L)$ and the elements $x \in L$ that are greater than $first(L)$. The recursive application of this "divide-and-conquer" strategy yields smaller and smaller partial lists or empty lists which need not be sorted. Finally, the program (1) generates a sorted list containing all elements of L by successively appending all partial lists previously sorted.

The Quicksort program (1) can also be constructed on the basis of elementary knowledge about the functions it contains. Roughly speaking, this knowledge need merely give the domains and ranges of the functions, that is, the sets on which the functions are defined and the sets of values the functions may take on. For example, the proposition $x < first(L)$ in (1) can be constructed on the basis of the knowledge that $first(L)$ is an element of L and $x < y$ is a proposition for any elements $x \in L$ and $y \in L$. An efficient selection of the functions used in the Quicksort program (1) seems feasible because they are very elementary such as the *append* function or even explicitly contained in the programming task such as the order relation "$<$".

For example, the SHUNYATA program generated theorem-proving programs by analyzing proofs of simple theorems on the basis of elementary knowledge about the functions the programs are composed of [1]. The development of these programs illustrates very simple aspects of the principles given above, for example, the unification of analytical spaces and the generation of economical variations. The theorem-proving programs developed by SHUNYATA generated proofs of further theorems, in particular, a proof of SAM's Lemma which is simpler than any proof known before. The complexity of SAM's Lemma more or less represented the state of the art in automated theorem proving [1, p. 561].

Gödel's incompleteness theorem says that every formal number theory contains an undecidable formula, that is, neither the formula nor its negation

are provable in the theory. The main problem in the proof of Gödel's theorem is the construction of such a formula. Analogously to the construction of the Quicksort program (1), an undecidable formula can be constructed on the basis of elementary rules for the formation of formulas, that is, on the basis of elementary knowledge about the symbols the formula contains. Ammon [2] describes a proof of Gödel's theorem and refers to further experiments with the SHUNYATA program.

4   Discussion

The Internet is a network of a varying number of computer programs and systems. My simple incompleteness theorem implies that no predefined formal system can completely model such a network because a program (computable function) not "captured" by the formal system can be constructed and added to the network. The proof of my theorem implies that the construction of this program can be achieved automatically.

Systems capable of communicating and interacting with humans more naturally than existing systems should be capable of reconstructing computable functions implicitly used in human forms of communication. My theorem implies that there is no predefined Turing machine or formal system modeling this communication.

Computer programs must be tested and debugged before they can be used in practice. My theorem implies that there is no general predefined algorithm for the verification of programs because a program (computable function) not "captured" by the algorithm can be constructed from the algorithm itself. Rather, the verification of programs is achieved on the basis of experience.

The Turing machine producing a sequence of computable functions f_1, f_2, ... in the proof of my incompleteness theorem can be regarded as an analytical space. The construction of another Turing machine producing f_1, f_2, ... and a computable function g not produced by the original machine can be regarded as the construction of a new analytical space. Thus, my simple theorem implies that all analytical spaces cannot be unified into a single analytical space. Roughly speaking, the development of new knowledge in analytical spaces cannot be regarded as a completely describable "closed box". Rather, it is an open process which may transcend any frame specified in advance.

My work can also be regarded as an investigation with the aid of com-

puters why Hilbert's decision problem (Entscheidungsproblem) is unsolvable because creative systems can determine beyond the limits of any predefined algorithm whether or not a statement in predicate calculus can be proved.

Church's thesis states that every effectively calculable function is general recursive [5, pp. 317-323]. Turing's thesis, which is equivalent to Church's thesis, states that every function that would be naturally regarded as computable is computable under his definition, that is, by a Turing machine [5, pp. 376-381]. My work should not be regarded as a refutation of Church's or Turing's thesis. Rather, it sheds new light on these theses and on Hilbert's decision problem (Entscheidungsproblem). In particular, it proves the existence of systems "computing" non-computable functions if "computing" means determining values of functions for given arguments. But these systems have no complete finite descriptions that can be given in advance. Rather, they can transcend any predefined formal description.

5   Related Work

Turing [9, 10] discusses whether "it is possible for machinery to show intelligent behaviour". Referring to Gödel's theorem and other, in some respects similar, results due to Church, Kleene, Rosser, and himself, Turing [9, p. 445] writes "that there are limitations to the powers of any particular machine". Turing [10, p. 4] states:

> The argument from Gödel's and other theorems ... rests essentially on the condition that the machine must not make errors. But this is not a requirement for intelligence.

He argues that a "man provided with paper, pencil, and rubber, and subject to strict discipline" can "produce the effect of a computing machine", but "discipline is certainly not enough in itself to produce intelligence" [10, p. 9 and p. 21]. My simple incompleteness theorem confirms that the powers of any particular Turing machine are limited. The theorem implies that systems capable of proving the theorem and interacting with their environment cannot be modeled by any predefined Turing machine. Thus, "ordinary computational systems" suitably equipped to prove the theorem and to interact with their environment can in principle transcend the powers of any machine completely specified in advance. My theorem implies that such systems are necessarily empirical and fallible in the sense that a complete predefined formalization of their truth judgments, for example, whether a given computer

program represents a computable function on the set of natural numbers, is impossible.

My theorem and principles about creative systems are compatible with Post's view [6, p. 417]:

> ... logic must ... in its very operation be informal. Better still, we write

> The Logical Process is Essentially Creative.

Wegner [11] argues that "interaction is more powerful than algorithms". My simple incompleteness theorem can be regarded as a mathematical proof of his thesis. Moreover, the theorem and its proof imply the existence of systems "computing" non-computable functions, that is, determining values of non-computable functions for given arguments. In this sense, they confirm the view of Wegner and Goldin [12] "that neither logic nor algorithms can completely model computing and human thought."

Wegner [11] writes that the incompleteness of interaction machines follows from the fact that dynamically generated input streams are mathematically modeled by infinite sequences over a finite alphabet, which are not enumerable. The "incompleteness", rather indeterminacy of creative systems follows from their definition, in particular from the fact that no formal theory can "capture" all computable functions on the set of natural numbers.

My incompleteness theorem implies that there is no general algorithm or formal system for the verification of programs. This result is compatible with Wegner's view "that proving correctness is not merely hard but impossible" because "*open, empirical, falsifiable, or interactive systems are necessarily incomplete*" [11, p. 10]. It is also compatible with Gödel's statement [3, p. 84] that "one has been able to define them [demonstrability and definability] only relative to a given language", that is, there is no general definition of formal proofs but such definitions can only be given in particular formal systems.

References

[1] Ammon, K. The automatic acquisition of proof methods. Proceedings of the Seventh National Conference on Artificial Intelligence, St. Paul, U.S.A, August 1988. Morgan Kaufmann, San Mateo, Calif., USA.

[2] Ammon, K. An automatic proof of Gödel's incompleteness theorem. Artificial Intelligence 61, 1993, pp. 291–306. Elsevier Science Publishers (North-Holland), Amsterdam.

[3] Gödel, K. Remarks Before the Princeton Bicentennial Conference on Problems in Mathematics. In M. Davis (Ed.), The Undecidable, Raven Press, New York, 1965.

[4] Gödel, K., On undecidable propositions of formal mathematical systems - POSTSCRIPTUM. In M. Davis, The Undecidable, Raven, Press, New York, 1965.

[5] Kleene, S. C. Introduction to Metamathematics. Wolters-Noordhoff, Groningen, and North-Holland, Amsterdam, 1952.

[6] Post, E. Absolutely Unsolvable Problems and Relatively Undecidable Propositions - Account of an Anticipation. In: M. Davis (Ed.), The Undecidable, Raven Press, New York, 1965, pp. 338–433

[7] Prasse, M., and Rittgen, P. Why Church's Thesis Still Holds. Some Notes on Peter Wegner's Tracts on Interaction and Computability. The Computer Journal, Vol 41, No. 6, 1998.

[8] Turing, A. M. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, Ser. 2, Vol. 42, 1936-37, pp. 230–265. Correction, *ibid.*, Vol. 43, 1937, pp. 544–546.

[9] Turing, A. M. Computing Machinery and Intelligence. Mind 59, No. 236, 1950, pp. 433–460.

[10] Turing, A. M. Intelligent Machinery. In: B. Meltzer and D. Michie (Eds.), Machine Intelligence 5, Edinburgh University Press, Edinburgh, 1969, pp. 3–23.

[11] Wegner, P. Why Interaction is More Powerful than Algorithms, Communications of the ACM 40 (5), 1997.

[12] Wegner, P., and Goldin, D. Computation Beyond Turing Machines, Communications of the ACM 46 (4), 2003.