

Why Church's Thesis Still Holds

Some Notes on Peter Wegner's Tracts on Interaction and Computability

Michael Prasse, Peter Rittgen

Institute of Business Informatics
University Koblenz-Landau
Rheinau 1, D-56075 Koblenz

email: {prasse, rittgen}@uni-koblenz.de

Abstract

Peter Wegner's definition of computability differs markedly from the classical term as established by Church, Kleene, Markov, Post, Turing et al.. Wegner identifies interaction as the main feature of today's systems which is lacking in the classical treatment of computability. We compare the different approaches and argue whether or not Wegner's criticism is appropriate. Taking into account the major arguments from the literature, we will show that Church's thesis still holds.

Introduction

In his 1997 paper, Wegner considers the relation between computability and interaction. He introduces a new machine model, namely the interaction machine, stating that the concept of interaction is so powerful that it challenges Church's thesis¹. Wegner also proposes a shift in the view of software development from a rational to an empirical perspective based on interaction.

Our paper tries to shed some new light on the ideas in [1] and related publications focusing on the computational power of Wegner's interaction machines and abstracting from the philosophical side of the matter. The article is divided into four sections: first we sketch Wegner's ideas. Then we give an overview of the classical framework of computability theory. We

proceed to the subject of interaction as introduced informally by Wegner and suggest formal models for his interaction machines. We show that such models cannot exceed the computational power of Turing machines and hence do not affect the validity of Church's thesis. We conclude by discussing the relevance of Wegner's work in the light of these results.

Wegner's Ideas

Wegner derives his ideas by observing that systems usually consist of communicating components: computers interface with the user, modern software is built from modules communicating with others, open systems react to external events in the environment, and the object-oriented programming paradigm describes software systems as nets of interacting objects. In this context, Wegner tries to harmonize interactive capabilities with computability theory but found that "Interactive tasks, like driving home from work, cannot be realized through algorithms. Algorithms that execute automatically without taking notice of their surroundings cannot handle traffic and other interactive events. [...] Interaction can simplify tasks when algorithms exist and is the only game in town for inherently interactive tasks, like driving or reserving a seat on an airline." [1, p. 82]

For such interactive tasks, he introduces interaction machines, i. e. Turing machines with input and output allowing dynamic interaction with the environment. This turns Turing machines into open systems. Wegner argues that these interaction machines are computationally more powerful than Turing machines. He distinguishes several types of interaction machines according to ways of input and capabilities. The most simple interaction machine is the interactive identical machine which outputs the input unchanged:

"P = in(message).out(message).P"

¹ also referred to as "Church-Turing thesis"

Wegner compares interaction to concurrency and distribution with respect to their expressive power and he comes to the conclusion that the latter do not give Turing machines greater power but the former does. He describes the behavior of an interaction machine in terms of the interfaces which offer the services of the machine to the environment, thus allowing the construction of systems from interactive components.

Finally, Wegner calls for a shift in the view of software development from a rational perspective towards an empirical approach, since Turing machines do not model important aspects of the real world such as interaction and real-time.

Computability Theory

The Algorithm

In computer science, an algorithm is understood as a procedure precisely describing in finite form the transformation of given input data into defined output data where the result is fully determined by the input². This also covers probabilistic and non-deterministic algorithms as long as the output is still determined uniquely. Uniqueness of output is always guaranteed by deterministic algorithms where at each point in time the next step of computation is given uniquely. Then the algorithm is a "recipe" of computation satisfying three conditions:

1. Only a finite number of steps (instructions) may be specified.
2. The first and last instruction can be identified uniquely; the effect of every step is defined, and so is its successor.
3. Every instruction can be carried out in the given context.

This definition of algorithm is closely linked to that of a function because an input (arguments) is transformed into an output (function value) justifying the term "algorithmic definition of a function". Consequently, a function is called

² cf. [2]

sequently, a function is called computable if it possesses an algorithmic definition. According to computability theory, there are uncountably many functions that are not computable.

Computability and Church's Thesis

A function³ is called computable if there is an algorithm computing the function value for any argument⁴. If the function is partial, the undefined case is represented by non-termination (no output, hence no result). To describe the class of computable functions more accurately, formal representations of algorithms have been developed such as Turing machines, Markov algorithms, flow charts, partial recursive functions and register machines. The classes of functions computed by these models are identical⁵.

Church's thesis generalizes this result to the statement that any function on positive integers (and hence on constructive objects⁶) computable by an algorithm in some intuitive sense is partial recursive. So the term "computability" is usually associated to the term "function"⁷. This is corroborated by the fact that partial recursive functions are sometimes used in defining the term "algorithm".

Other Procedures of Algorithmic Nature

There are some procedures resembling algorithms that do not fulfil all of the con-

³ The original works of Turing and Church refer to functions on the natural numbers but the results extend to general word functions $f: A^* \rightarrow B^*$ where A^* is the Kleene closure over alphabet A , or to the even more general functions on constructive objects (cf. footnote 6).

⁴ cf. [3]

⁵ Some proofs can be found in [4].

⁶ A constructive object is an object that is recursively constructed from a finite set of initial objects [5].

⁷ cf. [6] and [4] using the term function in the formulation of Church's thesis.

ditions mentioned: for example, protocols, regulations, directions for use, infinite control sequences and (to some extent) even role plays.

While these procedures might be called "algorithms" only in a very broad sense, non-deterministic algorithms⁸ differ from the ones adhering to the strict definition above in only one respect: the next step of computation is drawn arbitrarily from a set of successor steps. But then, an input x can have multiple outputs y . So, a non-deterministic algorithm defines a binary relation⁹ by associating to x all outputs possibly resulting from applying the algorithm to input x . This leads to the definition of a "computable relation". Algorithms of this type cannot be computed by a Turing machine although the characteristic function of a relation can.

Church's thesis remains unaffected by these "algorithms" because they do not compute functions. But as soon as equivalent computable functions are associated with these procedures, they do not exceed Turing machines in expressive power. They simply do not fulfil some of the requirements for computing functions.

Further Interpretations of Church's Thesis

Other approaches try to capture the intuitive notion of computability completely with Turing machines¹⁰. It should be mentioned that these ideas are discussed rather controversially in Philosophy and Artificial Intelligence but they are nevertheless not relevant to the ideas outlined here because they extend the scope of the original thesis. For example, in the literature we often find concepts like "mundane procedures" [10], functions on

the real numbers [11] and non-functional procedures like systems [12] or human thinking [13]. Based on these procedures, the authors typically try to construct counterexamples to Church's thesis. But these procedures (see also section *Other Procedures of Algorithmic Nature*) do not fulfil some of the fundamental requirements for algorithms like finite execution, determinateness and uniqueness of output, and the restriction of input/output to constructive objects (e.g. integers, words over a finite alphabet). We want to emphasize once more that for the remaining sections it is more important to note that the term "computability" is associated with the term "function".

Nevertheless, Church's thesis is significant even in its strict interpretation because statements concerning the computability of functions also show important limits of the algorithmic tractability of problems and hence hint at their general tractability. Problems typically contain functional parts to which the thesis applies. However, we have to concede that their might be non-functional parts and that these might even be of great importance to real-world problems¹¹, and in this case Church's thesis is of no help. But even then the validity of the thesis is not touched.

As a consequence, it is certainly worthwhile to investigate similar theses covering a broader spectrum of procedures. But often enough we find any hypothesis of the form "Every X Y is Z " under the heading of Church's thesis which gives rise to a lot of confusion [11]¹². Here we will concentrate on the strict interpretation of the thesis which we consider as the only admissible interpretation.

⁸ cf. [7].

⁹ Non-deterministic algorithms can compute functions if all possible computations for any input x yield the same result. In this case, non-deterministic algorithms coincide with the (deterministic) algorithms defined above.

¹⁰ cf. [8] and [9].

¹¹ Perhaps it can even be shown that effectively computable procedures are of no importance for "human affairs" at all.

¹² cf. also [14] where theses similar to that of Church are studied concerning (quantum) mechanics, human brain, computers and thought etc.

Discussion

Interactive Computations and Systems

An interactive system is a system that interacts with the environment via its input/output behavior. The environment of the system is generally identified with the components which do not belong to it. Therefore, an interactive system can be referred to as an open system because it depends on external information to fulfil its task. By integrating the external resources into the system, the system no longer interacts with the environment and we get a new, closed system. So, the difference between open and closed systems "lies in the eye of the beholder". Still, it is useful in the design of a system.

Today, a large number of systems are interactive in nature, e. g. information systems, office applications, development environments, retrieval systems, dialog systems, network applications and distributed systems. As already mentioned, even objects can be considered as machines on their own interacting with other objects. Interaction is an important characteristic of object-oriented systems. We can distinguish an internal view, comprising interaction between objects within the system, from an external view of the behavior of the system and its user interface. Considering the existence of computer applications for such interactive systems, we face the question of how these systems relate to Turing machines.

Turing Machines without I/O

Turing machines only describe the process of computation. In conformity to a function, all inputs are given prior to computation. Upon termination, the tape contents are interpreted as output. So, Turing machines are closed systems without intermediate input/output describing functions.

Contrary to that, computers possess dedicated input/output channels to communi-

cate with the human user and the system environment, but their resources are limited (there is no infinite tape). In this respect, we have to acknowledge Wegner's achievements of having recognized the importance of additional input/output operations and investigating their influence on the performance of computers.

Relevance of Interaction

Fig. 1 shows the flow chart of a simple interactive program that repeatedly computes the square of natural numbers entered by the user.

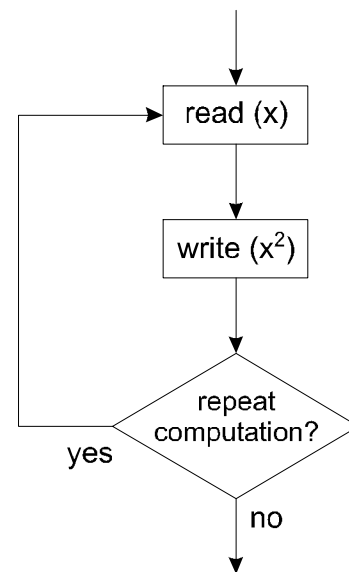


Fig. 1: A simple interaction

Regardless of the programming paradigm, interaction provides communication facilities to control the flow of the program. The given flow chart describes the procedure completely, uniquely and finitely. Neglecting input/output, each iteration can be interpreted as a computation performed by a Turing machine. But the influence of the (unknown) user input on the control flow makes it hard to determine which overall function is computed by this procedure (or if the procedure can be seen as a computation at all).

One solution might be to consider the combination of program plus user: Then, the corresponding function is

$f: N^* \rightarrow N^*$ where $f(x_1 x_2 x_3 \dots x_n) := x_1^2 x_2^2 x_3^2 \dots x_n^2$ and $f(\varepsilon)$ undefined¹³.

The input will be determined only at run-time. The overall function is derived by integrating the user into the computation, thus closing the system.

It is evident that Turing machines cannot model this behavior directly due to the missing input/output operations. Therefore, we need models that take into account inputs and outputs at run-time to describe computer systems adequately.

Interaction Machines

Wegner defines interaction machines as Turing machines with input and output operations, thus turning them into open systems depending on external factors. By "outsourcing" tasks to external partners, the computation requires the interplay of these partners. However, Wegner leaves open a series of questions:

- How is the introduction of input/output operations done?
- What do interaction machines compute (or accept)?
- How do you define the operation of such a machine?
- How do interaction and communication work?
- What is the output and how do you interpret it?

If you compare Wegner's characterization of an interaction machine to the formal definition of a Turing machine, it is obvious that the former cannot be accepted as a computational model. Moreover, Wegner does not provide a precise constructive example to support his thesis and to contradict Church's thesis. Hence, we can hardly assess the validity of his statements.

Formal Definitions of Interaction Machines

Wegner's idea to equip Turing machines with interaction capabilities can already be found in [15, p. 235 ff]. There, interactive proof systems and "Arthur-against-Merlin games" are treated based on interactive Turing machines with a well-defined behavior.

An interactive Turing machine (ITM) is a Turing machine with a read-only input tape, a read-only random tape, a working tape, a read-only communication tape, a write-only communication tape and a write-only output tape. An interactive protocol is an ordered pair (M_p, M_v) of ITMs sharing input and communication tapes (the latter with reversed access, see fig. 2).

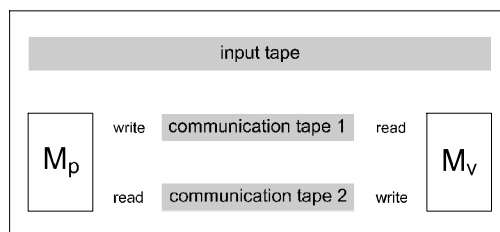


Fig. 2: Interactive protocol

The operation of a protocol consists of communication rounds starting with M_v (the verifier). After having sent a message to M_p (the prover), M_v is deactivated and M_p takes up its computation. A message from the prover to the verifier reverses the roles to the original status quo and finishes the first round. M_p suggests proofs while M_v accepts or rejects them.

A language L has an interactive proof system if, for a fixed verifier M_v , a prover M_p exists such that the protocol (M_p, M_v) accepts for each x in L , and for all other provers the corresponding protocol does not accept for each x not in L . The interactive proof itself proceeds along the following lines: The prover suggests a proof to the verifier (e. g. whether a word x is an element of language L). The verifier examines this proof. If the proof is convincing, the verifier accepts it and the

¹³ ε is the empty word.

ing, the verifier accepts it and the protocol stops with a positive answer.

We gave the example of the interactive protocols to show that it is possible in principle to specify interaction machines formally and hence exactly; nevertheless, interactive protocols cannot be related to Wegner's interaction machines due to the probabilistic nature of the former (and the complexity restriction). So in the sections to follow, we will base discussion on deterministic interactive Turing machines similar to those of [15]. We thus focus on the communication facilities offered by the communication tape(s) and not on the probabilistic features. However, we do not try to make up for the lacking formalization of Wegner's interaction machines.

Nevertheless, it should be mentioned that beyond interactive Turing machines many models exist using interaction or communication, like process algebras, Petri nets, neural networks, communication protocols, etc.

Interaction and Church's Thesis

After having prepared the ground, we can now investigate the relation between Church's thesis and interaction. Peter Wegner writes: "The hypothesis that the formal notion of computability by Turing machines corresponds to the intuitive notion of what is computable has been accepted as obviously true for 50 years. However, when the intuitive notion of what is computable is broadened to include interactive computations, Church's thesis breaks down. Though the thesis is valid in the narrow sense that Turing Machines express the behavior of algorithms, the broader assertion that algorithms capture the intuitive notion of what computers compute is invalid." [1, p. 83]

Wegner argues that interaction challenges the term "computability": Machines capable of input and output are more powerful than Turing machines. This statement requires the specification of a set of problems solvable by interaction machines and not solvable by Turing ma-

chines: Is there, for example, an interaction machine for solving the halting problem? Nevertheless, it is certainly very interesting to ask whether interaction machines can "do" anything Turing machines cannot. Investigating the expressive power of interaction machines can be done in two steps:

1. Are procedures which do not compute any function expressible by interaction machines? And if so, what does computability mean in the case of such tasks as "driving home"? Peter Wegner calls these computations "non-algorithmic" but he does not give a precise definition of the term.
2. Are there functions computable by interaction machines but not by Turing machines? This question is even more interesting than the first: If answered in the affirmative, we would have a model computing more functions than the Turing machine, and Church's thesis would indeed be contradicted. It is important to note that Wegner does not give any examples for this case in his publications.

To answer the above questions, we will view interaction machines from three different perspectives: First we investigate the additional computational power provided by input and output, secondly, the incompleteness of interaction machines and the resulting relative model of computation. Finally, we take a component-oriented look at interaction machines.

Input and Output

Interaction machines are defined as Turing machines with input and output. Therefore, their internal behavior and expressiveness do not differ from that of an equivalent Turing machine. Though Wegner leaves open the question of how the input/output mechanism works, it can be assumed that input and output only serve data transport without any computational capabilities. Therefore, the interaction machine itself does not possess greater computational power than a Turing machine. But through communication,

the computational capabilities of other machines can be utilized. Interaction can then be interpreted as a (subroutine) call.

Relative Computation Model

Assuming this view, interaction machines lead to a relative model of computation resembling that of oracle machines. Conversely, there are approaches which view oracles in terms of interactions e.g. [16], where the oracle machine interacts with the oracle. This shows the close relation between interaction machines and oracle machines.

Wegner employs the following "proof" to show that interaction machines are more expressive than Turing machines: "Interaction machines that passively interact with input sequences generated by oracles or processes in nature can have richer behavior in a more direct sense since their input may not have a recursively enumerable specification." [17, p. 28] This proof relies on a comparison with oracles. The greater computational power does not arise from the input/output operations but from the external interaction partners. Oracle machines compute the solution of a problem with respect to an oracle. Only if the oracle is undecidable, can oracle machines solve non-computable problems. A decidable oracle, however, coincides with a subroutine call to a Turing machine, so that there is a Turing machine for the complete problem which is thus computable.

But an oracle machine alone cannot compute a function¹⁴. It is like a template from which a concrete oracle machine is instantiated for a certain problem only if a specific oracle is defined. This can be transferred to interaction machines whose important characteristic is the indeterminateness of the output due to intermediate inputs. Contrary to oracle machines, where the oracle has to be specified, Wegner leaves this point open. So, the interaction machine of Wegner is also

merely a template. A concrete machine does not result until we specify interaction partners and interaction behavior.

Therefore, interaction machines lead to a relative model of computation. Computations by these machines have to be regarded relative to the interaction participants. If we assume that the interaction partners of such a machine can only treat computable problems, then the resulting machine can also only solve computable problems.

Interaction Machines as Components

Interaction is essentially only a new way of putting together procedures. The partial recursive functions are closed with respect to this operation (like Turing machine sequencing) because any interaction can be simulated by a subroutine. Due to its incomplete specification and dependency on interaction partners, it cannot in general compute a function on its own. So, interaction machines are not algorithms according to computability theory but components of interactive systems. These systems (protocols), and not the interaction machines themselves, define a computation model with respect to certain interaction process and accepting rule. The protocols specify which type of procedures are performed by interaction machines. This means that, independent from the interaction machine, the protocol decides whether the procedure computed is an algorithm or not. To compare interaction machines with Turing machines, the chosen protocol has to calculate functions.

If human beings or external processes act as interaction partners, further problems concerning the computability arise because it is immediately evident that they all have to behave partial recursively for the overall procedure to be computable. It is a well-known fact that a system with a non-computable component can itself lose computability. But this is not particular to interaction: Any model of computability providing composition possesses this feature.

¹⁴ We abstract from the case of an oracle machine which does not call its oracle (simple Turing machine).

Conclusion

From these three views on interaction machines, we conclude that models based on a set of partial recursive interaction machines are not capable of computing non-computable functions. So, interaction does not lead to an enhanced computability of functions. Fig. 3 depicts the relation between the problem classes. Please note that the set of tasks definable by interaction machines which do not represent partial recursive functions is potentially empty.

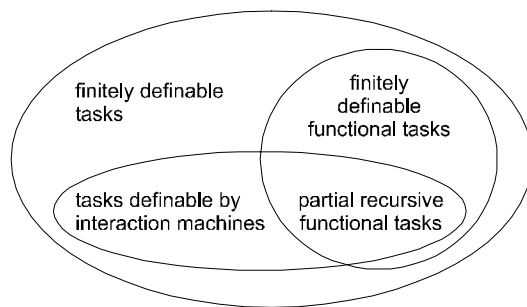


Fig. 3: Computational power of interaction machines

Wegner himself must have recognized some of the objections mentioned here because he restricted his original claim in later publications: In [17, p. 2] he boldly asserts: "Interaction machines, defined by extending Turing machines with input actions (read statements), are shown to be more expressive than computable functions, providing a counterexample to the hypothesis of Church and Turing that the intuitive notion of computation corresponds to formal computability by Turing machines." One year later, he states more cautiously: "Though Turing machines capture the intuitive semantics of algorithms, the broader Church-Turing thesis (that Turing machines capture the intuitive notion of computing) is invalid" [18, p. 1].

We can draw the conclusion that Wegner's computability focuses more on real computers and systems than on the theoretical, function-oriented view of Church, Kleene, Markov, Post and Turing.

Synthesis

Peter Wegner raises an interesting problem for both theoretical computer science and software development: How does interaction fit into computability theory? This question and its possible answers are certainly fascinating for every computer scientist. But nevertheless we do not agree with Wegner's conclusion that interaction contradicts Church's thesis.

Peter Wegner draws this conclusion on the basis of a broader understanding of computability than that of Church (i. e. the computability of functions). His interaction machines cannot compute non-recursive functions, so Church's thesis still holds.

To characterize Wegner's ideas better, we should rather speak of "computerability" than computability. While the latter deals with (mathematical) computations, the former focuses on computer applications which include important aspects of real-world systems like interaction. So, a "computer-able" model is one that suits the description of such systems. Wegner claims that his interaction machines have this property.

If we consider Wegner's work in this light, then it must be conceded that he has contributed an important approach to the description of modern software systems with interactive user interfaces. These aspects are certainly not covered by the concept of a Turing machine. Interaction machines, however, allow a fair description of object-oriented, modular systems. Together with other methods¹⁵, they offer an interesting way of formally describing object-oriented systems¹⁶.

¹⁵ e. g. "programming by contract" [19]

¹⁶ Although a formal definition of interaction machines still remains to be given.

References

- [1] Wegner, P. (1997) Why interaction is more powerful than algorithms. *CACM*, **40 (5)**, 80-91.
- [2] Uspenskii, V. A. (1987) Algorithm. In Hazewinkel, M. (ed) *Encyclopaedia of Mathematics*. Kluwer Academic Publishers, Dordrecht.
- [3] Lavrov, I. A. and Taimanov, A. D. (1987) Computable Function. In Hazewinkel, M. (ed) *Encyclopaedia of Mathematics*. Kluwer Academic Publishers, Dordrecht.
- [4] Hopcroft, J. E. and Ullman, J. D. (1979) *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, Reading MA.
- [5] Nagorny, N. M. (1987) Constructive Object. In Hazewinkel, M. *Encyclopaedia of Mathematics*. Kluwer Academic Publishers, Dordrecht.
- [6] Adyan, S. I. (1987) Church Thesis. In Hazewinkel, M., *Encyclopaedia of Mathematics*. Kluwer Academic Publishers, Dordrecht.
- [7] Loeckx, J. (1976) *Algorithmentheorie*. Springer, Berlin.
- [8] Copeland, J. (1996) The Church-Turing Thesis. In: *Online Stanford Encyclopedia of Philosophy*. <http://plato.stanford.edu>.
- [9] Kearns, J. T. (1997) Thinking Machines: Some Fundamental Confusions. *Minds and Machines*, **7 (2)**, 269-287.
- [10] Cleland, C. E. (1993) Is the Church-Turing Thesis True? *Minds and Machines*, **3 (3)**, 283-312.
- [11] Horsten, L. and Roelants, H. (1995) The Church-Turing Thesis and Effective Mundane Procedures. *Minds and Machines*, **5 (1)**, 1-8.
- [12] Shagrir, O. (1997) Two Dogmas of Computationalism. *Minds and Machines*, **7 (3)**, 321-344.
- [13] Fetzer, J. H. (1997) Thinking and Computing: Computers as Special Kinds of Signs. *Minds and Machines*, **7 (3)**, 345-364.
- [14] Odifreddi, P. (1989) *Classical Recursion Theory*. Elsevier Science Publishers, Amsterdam.
- [15] Balcazar, J. L., Diaz, J. and Gabarro, J. (1990) *Structural Complexity II*. Springer, Berlin.
- [16] Schöning, U. (1988) Complexity Theory and Interaction. In Herken, R. (ed) *The Universal Turing Machine – A Half-Century Survey*. Springer, Berlin.
- [17] Wegner, P. (1995) Tutorial Notes: Models and Paradigms of Interaction. *OOPSLA*, <http://www.cs.brown.edu/people/pw/>.
- [18] Wegner, P. (1996) Interactive Foundations of Computing. Unpublished manuscript, December, <http://www.cs.brown.edu/people/pw/>.
- [19] Meyer, B. (1997) *Object-Oriented Software Construction*. 2nd Edition, Prentice Hall, Englewood Cliffs.