

MICHAEL PRASSE  
PETER RITTGEN

# Why Church's Thesis Still Holds

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

Dezember 1997

---



MICHAEL PRASSE  
PETER RITTGEN

# Why Church's Thesis Still Holds

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

Dezember 1997

---

Die Arbeitsberichte des Instituts für Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i.d.R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar.

The "Arbeitsberichte des Instituts für Wirtschaftsinformatik" comprise preliminary results which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

---

Alle Rechte vorbehalten. Insbesondere die der Übersetzung, des Nachdruckes, des Vortrags, der Entnahme von Abbildungen und Tabellen - auch bei nur auszugsweiser Verwertung.

All rights reserved. No part of this report may be reproduced by any means, or translated.

---

**Anschrift der Verfasser**  
**Address of the authors:**

Dipl. Inf. Michael Prasse  
Dr. Peter Rittgen  
Institut für Wirtschaftsinformatik  
Universität Koblenz-Landau  
Rheinau 1  
D-56075 Koblenz

**Arbeitsberichte des Instituts für  
Wirtschaftsinformatik**  
**Herausgegeben von / Edited by:**

Prof. Dr. Ulrich Frank  
Prof. Dr. J. Felix Hampe

©IWI 97

---

**Bezugsquelle / Source of Supply:**

Institut für Wirtschaftsinformatik  
Universität Koblenz-Landau  
Rheinau 1  
56075 Koblenz  
Tel.: 0261-9119-480  
Fax: 0261-9119-487  
Email: [iwi@uni-koblenz.de](mailto:iwi@uni-koblenz.de)  
WWW: <http://www.uni-koblenz.de/~iwi>



## **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 lacking in the classical treatment of computability. We compare the different approaches and argue whether or not Wegner's criticism is appropriate.



## Introduction

Wegner 1997 considers the relation between computability and interaction. It introduces a new machine model, namely the interaction machine, stating that the concept of interaction is so powerful that it challenges Church's Thesis. Based on interaction, Wegner also proposes a change in view of software development from a rational to an empirical perspective.

Our paper tries to shed some new light on the ideas in Wegner 1997 and other publications.

## Wegner's Ideas

Wegner derives his ideas from 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, the object-oriented programming paradigm describes software systems as nets of interacting objects. In this context, Wegner tried to harmonize the 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.“ [Wegner 1997, p. 82]

For such interactive tasks, he introduced 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 regarding ways of input and capabilities. The most simple interaction machine is the interactive identical machine which outputs the input unchanged:

$$„P = in(message).out(message).P“$$

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 demands to shift the view of software development from a rational perspective towards an empirical approach because Turing machines do not model important aspects of the real world, like interaction and real-time.

## Computability Theory

### *The Algorithm*

In computer science, the algorithm is understood as a precisely defined procedure describing in finite form the transformation of given input data into defined output data. At each point in time, the next step of computation can be determined uniquely. So, the algorithm is a „recipe“ of computation satisfying three conditions [Loeckx 1976, p. 7 ff]<sup>1</sup>:

---

<sup>1</sup> Similar definitions can be found in Börger 1989, p. 3 f, Gellert et al. 1971, p. 779 ff, Herschel 1974, p. 157 ff, Lew 1985, p. 25, Sommerhalder and van Westrenen 1988, p. 34, Uspenskii 1987 and Wood 1987, p. 56 ff.

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 precisely 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 computable if it possesses an algorithmic definition. According to computability theory, there is an uncountably infinite amount of functions that are not computable.

### ***Computability and Church’s Thesis***

A function is called computable if there is an algorithm computing the function value for any argument [Loeckx 1976, p. 9]<sup>2</sup>. 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, partially recursive functions and register machines. The classes of functions computed by these formalisms are identical<sup>3</sup>.

Church’s Thesis generalizes this result to the statement that any function computable by an algorithm in some intuitive sense is partially recursive. An equivalent notion can be found in Gurari 1989 with Turing machines as the reference: „Church’s Thesis: A function is computable (respectively, partially computable) if and only if it is computable (respectively, partially computable) by a deterministic Turing transducer“<sup>4</sup>.

So, the term „computability“ is usually associated to the term „function“<sup>5</sup>. This is corroborated by the fact that partially 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 conditions mentioned: for example, non-deterministic algorithms [Loeckx 1976] where the next step of computation is drawn arbitrarily from a set of successor steps. A non-deterministic algorithm therefore defines a binary relation<sup>6</sup> by associating all possible outputs to input  $x$ :

{  $y$  |  $y$  is a possible result of applying the algorithm to input  $x$  }

Generally, non-deterministic algorithms do not possess more expressive power than deterministic ones<sup>7</sup>. So, Church’s Thesis remains unaffected by non-determinism.

---

<sup>2</sup> cf. also Lavrov and Taimanov 1987, Lew 1985, p. 77, Schöning 1992, p. 85 f, and Wood 1987, p. 58 ff

<sup>3</sup> Proofs can be found in Börger 1989, Bridges 1994 and Hopcroft and Ullman 1990.

<sup>4</sup> Other approaches capture the intuitive notions of computability and procedure completely by Turing machines [Gellert et al. 1990].

<sup>5</sup> cf. Adyan 1987, Bridges 1994, Hopcroft and Ullman 1990, p. 178, and Marchenkov and Nagorny 1987 all using the term function in the formulation of Church’s Thesis

<sup>6</sup> Non-deterministic algorithms can compute functions if all possible computations for any input  $x$  yield the same result.

<sup>7</sup> Loeckx’ definition of non-deterministic algorithms differs from the common one in that Loeckx uses the whole tree of computations instead of the usual path to define the relation. Consequently, his non-deterministic Turing machine (contrary to a common one) cannot be simulated by a deterministic Turing machine.

Other procedures of algorithmic nature, like protocols, regulations, directions (for use), infinite control sequences and (to some extent) role plays, do not even challenge Church's Thesis because they do not describe functions (although functions may be attributed to them).

## 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 whether such systems can still be modelled by Turing machines.

### *Turing Machines without Input/Output*

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 mathematical functions.

Contrary to that, computers possess dedicated input/output channels to communicate 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.

### *Semantics of Interaction*

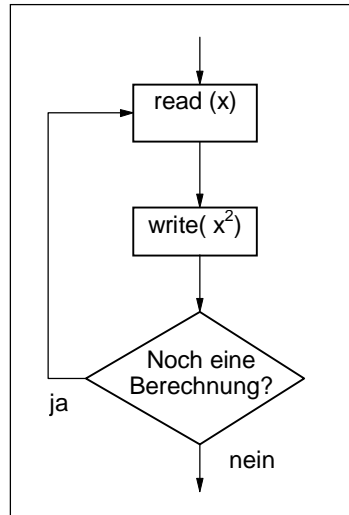
Fig. 1 shows the flow chart of a simple interactive program that repeatedly computes the square of natural numbers entered by the user.

The given flow chart describes the procedure completely, uniquely and finitely, so the conditions for an algorithm are met. But, the influence of the (unknown) user input on the control flow makes it hard to determine which function is computed by this algorithm (or if the algorithm 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

$$f: N^* \rightarrow N^* \text{ where } f(x_1 x_2 x_3 \dots x_n) := x_1^2 x_2^2 x_3^2 \dots x_n^2.$$

The input is determined at run-time and the user is integrated into the computation, thus closing the system.



**Fig. 1: A simple interaction**

Another example for interaction is an operating system. If we restrict its task to executing programs (leaving out file, user and process management, and resource limitations), the operating system can be seen as an interactive loop according to fig. 1. It can be implemented by a universal Turing machine using gödelization: This machine receives a number  $k$  and an input  $x$ , and runs the machine  $k$  in the gödelization with input  $x$ . The input in fig. 1 asks the user for  $k$  and  $x$  (which program do you want to run with which data?). The function computed by this operating system (including the user) would be:

$$f: (N \times \Sigma^*)^* \rightarrow \Sigma^* \text{ mit } f((n_1, x_1)(n_2, x_2) \dots (n_k, x_k)) := \begin{cases} n.d. & \text{falls } \bigvee_{i \in \{1 \dots k\}} (x_i \notin D_{n_i}) \\ \varphi_{n_1}(x_1) \varphi_{n_2}(x_2) \dots \varphi_{n_k}(x_k) & \text{falls } \bigwedge_{i \in \{1 \dots k\}} (x_i \in D_{n_i}) \end{cases}$$

This function reveals an interesting problem concerning the output: If one of the subfunctions is undefined for a given argument, the corresponding submachine does not halt on this input, thus stopping the whole machine computing  $f$  and producing no output at all, whereas a combination of interaction machine and user would at least output the partial results up to this point. But even the interaction machine cannot terminate the remaining computations.

It is understood that Turing machines cannot model this behavior of subsystems 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 accurately.

### **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 does input/output work?
- 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, the lack of formality in Wegner’s approach makes the validation of his theses nearly impossible.

Wegner argues that interaction challenges the term „computability“: Machines being 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 machines): Is there, for example, an interaction machine solving the halting problem?

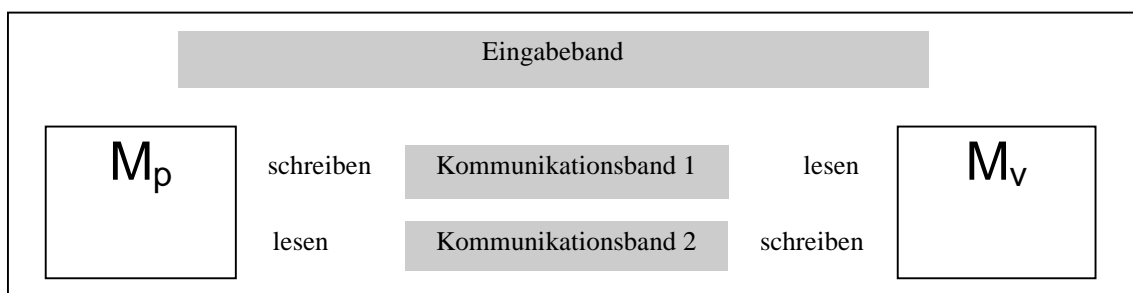
An important characteristic of interaction machines is the indeterminacy of the output because intermediate inputs can change the output at all times. Due to its incomplete specification and dependency on interaction partners, it cannot in general compute a function on its own. So, interaction machines are no algorithms according to computability theory but components of interactive systems. If human beings act as interaction partners, further problems concerning the computability arise.

The observable behavior of interaction machines can be described by interaction histories documenting the system behavior over time. The initial input and the interaction history determine uniquely the work of the interaction machine up to a certain point in time.

### ***Other Definitions of Interaction Machines***

Wegner’s idea to equip Turing machines with interaction capabilities can already be found in Balcazar et al. 1990, 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. After an appropriate amount of rounds, final acceptance is done by  $M_v$ .

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 protocol stops with a positive answer.

The example of the interactive protocols shows that a precise definition of interaction machines is possible.

### ***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.“ [Wegner 1997, p. 83]

Church's Thesis, however, does not assert that algorithms describe any behavior of computers but only that algorithms can compute any computable function. So, Wegner's argument does not affect Church's Thesis.

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 not computing any function expressable 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 it could be answered in the affirmative, we would have a computational model that is more expressive than the Turing machine, and Church's Thesis would be contradicted. It is important to note that Wegner does not give any examples for this case in his publications. We cannot answer the question here because his interaction machines are not formally defined. But if we assume the interaction machine given above, interactive protocols<sup>8</sup> do not exceed the expressive power of Turing machines.

Interaction machines are defined as Turing machines with input and output. Therefore, their internal behavior and expressiveness does not differ from that of an equivalent Turing machine. Though Wegner leaves open how the input / output mechanism works, it can be assumed that input and output only serve data transport without any computational capabilities. But through communication, the computational capabilities of other machines can be utilized. Interaction can then be interpreted as a (subroutine) call. Assuming this view, interaction machines resemble oracle machines [Wagner and Wechsung 1986, p. 48 f].

Oracle machines compute the solution of a problem with respect to an oracle. Only if the oracle is undecidable, oracle machines can 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.

If we assume that the interaction partners of an interaction machine can treat computable problems only, then the resulting machine can also solve computable problems only! 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

---

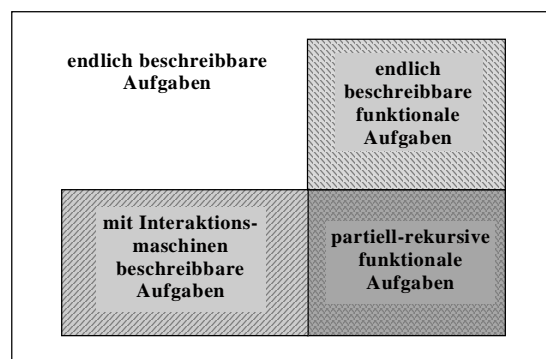
<sup>8</sup> The interaction machine itself does not compute a function because it is open with respect to input and output. Only for a closed system of such machines (i. e. a protocol), the computed function can be defined meaningfully.

enumerable specification.“ [Wegner 1995, 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. Contrary to the oracle machines where the oracle has to be specified, Wegner leaves this point open. In this light, the interaction machine of Wegner is more a template than a concrete machine for the solution of a problem.

So, interaction is essentially only a new way of putting together procedures. The partially-recursive functions are closed with respect to this operation (like Turing machine sequencing) because any interaction can be simulated by a subroutine. If persons or external processes are included in the interaction, it is immediately evident that they all have to behave partially-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. For this reason, undecidable oracles are usually excluded from consideration because the solution of such oracles is not effective<sup>9</sup> and it cannot be computed by an interaction machine<sup>10</sup>.

Along these lines, even interaction machines are not capable of computing non-computable functions because they possess the same functional expressiveness as Turing machines. 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 not representing partially-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. So, in Wegner 1995, 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 carefully: „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“ [Wegner 1996, 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.

<sup>9</sup> An effective problem is one that can be computed by an algorithm.

<sup>10</sup> An undecidable oracle can be allowed using mechanisms analogous to that of oracle machines.

## Finite Description

Wegner describes interaction machines as Turing machines enhanced by input and output. Though he does not say how this enhancement is done, we can assume that the fundamental requirement of a finite description of the procedure is met. Otherwise, interaction machines could not be specified.

Consider the example of „driving home without an accident“<sup>11</sup>. An interaction machine for this task may react to events concerning the traffic. Due to the finiteness of the description, the amount of reactions to the events is finite as well. This means that the events can be partitioned into equivalence classes with respect to their treatment. All events that are not considered belong to the class of non-treatable events which can lead to failure of the machine. Therefore, Peter Wegner’s interaction machines cannot guarantee arriving at home safely because events may occur to which no reaction has been specified.

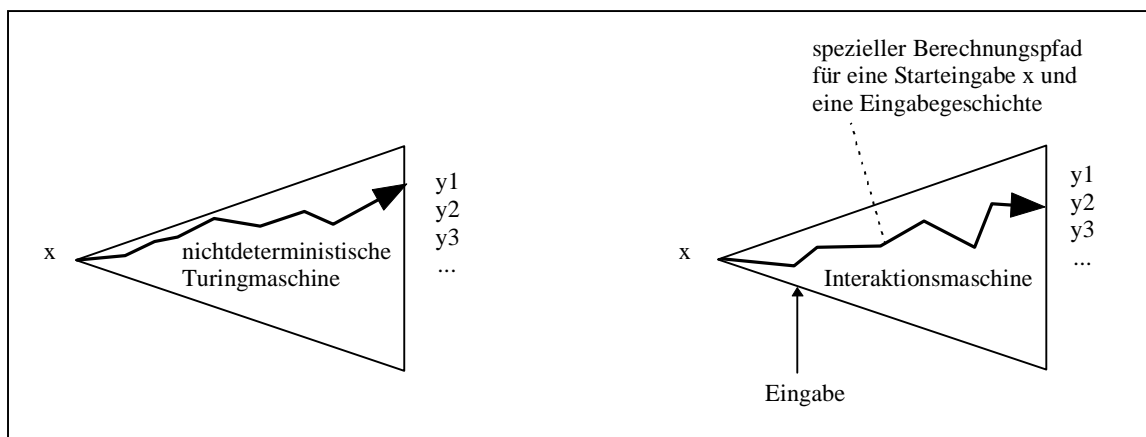
It has to be assumed that „driving home“ is a task that cannot be described finitely and completely due to the uncertainty of events and the dynamics of the task. To put it differently: The completeness and adequacy of a formal description of this task can hardly be proved.

Using abstraction and simplification, a real system can often be described by a finite interactive model. This model is then formal and complete, but it does not necessarily describe the reality completely and accurately!

## Interaction and Non-determinism

Similar to non-deterministic Turing machines<sup>12</sup>, the computation of an interaction machine can be seen as a possibly infinite tree where the path of computations is determined by run-time inputs.

Fig. 4 compares the computation paths of interaction machines and non-deterministic Turing machines. While non-deterministic Turing machines arbitrarily choose the next step from a set of instructions, interaction machines choose it depending on the possible run-time inputs. In both cases, the set of successor instructions is finite. This follows immediately from the requirement of a finite description.



**Fig. 4: Computation paths of interaction machines**

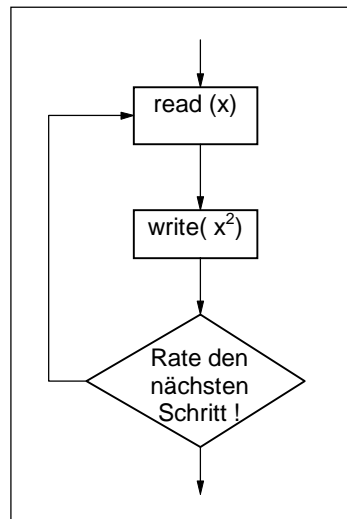
Neglecting time issues, we can simulate the functioning of an interaction machine by a non-deterministic Turing machine. Whenever the interaction machine expects an input, the non-deterministic Turing machine invokes a computation guessing the input in a non-deterministic way. After that, the interaction machine proceeds with its computations.

Fig. 5 shows how the interaction of fig. 1 can be replaced by non-determinism. The interaction with the user is simulated by guessing. Because the input has to be finite to be processed, the non-deterministic Turing machine can guess it. In case of infinite inputs, the guessing step would never terminate.

<sup>11</sup> cf. Wegner 1997

<sup>12</sup> We assume non-deterministic Turing machines to compute relations.

So, interaction can be reduced to a form of non-determinism<sup>13</sup>.



**Fig. 5: Interaction and non-determinism**

## Synthesis

Peter Wegner raised an interesting problem for both theoretical computer science and software development: How does interaction fit into computability theory? But we do not approve of his conclusion that Church's Thesis is contradicted.

Peter Wegner draws this conclusion with a different understanding of computability exceeding that of Church (i. e. the computability of functions). His interaction machines cannot compute non-recursive functions, so Church's Thesis still holds.

Peter Wegner's understanding of computability might be better characterized by the term „system adequacy“: In our opinion, his claim can be paraphrased as „Turing machines do not describe systems (of humans and computers) adequately“, a fact to which we readily agree. Generally, a system cannot be reduced to a function (an algorithm).

If we consider Wegner's work in this light, he 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 methods<sup>14</sup>, they offer an interesting way of describing object-oriented systems formally<sup>15</sup>.

## References

- Adyan, S. I. (1987) Church Thesis. In Hazewinkel, M., *Encyclopaedia of Mathematics*. Kluwer Academic Publishers, Dordrecht.
- Balcazar, J. L., Diaz, J. and Gabarro, J. (1990) *Structural Complexity II*. Springer, Berlin.
- Börger, E. (1989) *Computability, Complexity, Logic*. North-Holland, Amsterdam.

<sup>13</sup> We obtain the same result using interactive protocols.

<sup>14</sup> e. g. „programming by contract,, [Meyer 1997]

<sup>15</sup> Although a formal definition of interaction machines still remains to be given.

- Bridges, D.S. (1994) *Computability - A Mathematical Sketchbook*. Springer, Berlin.
- Gellert, W., Küstner, H., Hellwich, M. and Kästner, H. (1971) *Kleine Enzyklopädie Mathematik*. VEB Bibliographisches Institut, Leipzig.
- Gellert, W., Kästner, H. and Neuber, S. (1990) *Lexikon der Mathematik*. VEB Bibliographisches Institut, Leipzig.
- Gurari, E. (1989) *An Introduction to the Theory of Computation*. Computer Science Press, Rockville MD.
- Herschel, R. (1974) *Einführung in die Theorie der Automaten, Sprachen und Algorithmen*. Oldenbourg, München.
- Hopcroft, J. E. and Ullman, J. D. (1990) *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*. Addison Wesley, Reading MA.
- Loeckx, J. (1976) *Algorithmtheorie*. Springer, Berlin.
- Lavrov, I. A. and Taimanov, A. D. (1987) Computable Function. In Hazewinkel, M. *Encyclopaedia of Mathematics*. Kluwer Academic Publishers, Dordrecht.
- Lew, A. (1985) *Computer Science: A Mathematical Introduction*. Prentice Hall, Englewood Cliffs.
- Marchenkov, S. S. and Nagornyi, N.M. (1987) Turing Machine. In Hazewinkel, M. *Encyclopaedia of Mathematics*. Kluwer Academic Publishers, Dordrecht.
- Meyer, B. (1997) *Object-Oriented Software Construction*. 2nd Edition, Prentice Hall, Englewood Cliffs.
- Schöning, U. (1992) *Theoretische Informatik kurzgefaßt*. BI-Wissenschaftsverlag, Mannheim.
- Sommerhalder, R. and van Westrenen, S. C. (1988) *The Theory of Computability: Programs, Machines, Effectiveness and Feasibility*. Addison Wesley, Reading MA.
- Uspenskii, V. A. (1987) Algorithm. In Hazewinkel, M. *Encyclopaedia of Mathematics*. Kluwer Academic Publishers, Dordrecht.
- Wagner, K. and Wechsung, G. (1986) *Computational Complexity*. VEB Deutscher Verlag der Wissenschaften, Berlin.
- Wegner, P. (1995) Tutorial Notes: Models and Paradigms of Interaction. *OOPSLA*, <http://www.cs.brown.edu/people/pw/>
- Wegner, P. (1996) Interactive Foundations of Computing. Unpublished manuscript, December, <http://www.cs.brown.edu/people/pw/>
- Wegner, P. (1997) Why interaction is more powerful than algorithms. *CACM*, **40** (5), 80-91.
- Wood, D. (1987) *Theory of Computation*. John Wiley & Sons, New York.