1. Introduction

1.1. Turing Machines and The Halting-Problem after [1]

Theorem: The halting problem can not be decided on a Turing machine.

1.2. Illustration after [2]

The question really is wether it is possible to write a program on a certain given machine *M*, which evaluates **all** programs on *M* in such a way wether they would end up in an infinite loop or terminate correctly.

A mind theory leads into a contradiction:

Given that on a UNIX-computer *U* exists a program ‚*InfLoop*' with the characteristics

> *InfLoop* accepts one commandline argument

and

> For all programs *P* on *U*, which do not accept a commandline argument, it holds
>
> 1. (*InfLoop P*) terminates correctly
> 2. (*InfLoop P* = 0) ⇔ (*P* terminates correctly)

Then one can construct a program ‚*Paradox*' on *U* , which leads the existence of ‚*InfLoop*' into a contradiction (C-Code of ‚Paradox' see next page 1.3.). With this program ‚*Paradox*' it holds

$$(InfLoop\ Paradox = 0) \quad \Leftrightarrow \quad (InfLoop\ Paradox \neq 0)$$

This is a contradiction!

1.3. ‚Paradox' in C-Code

```
#include <assert.h>
#include <errno.h>
#include <process.h>
#include <stdio.h>

#define FALSE NULL
#define TRUE !FALSE

#define INFLOOP "InfLoop"
#define PARADOX "Paradox"

extern int main(int argc, char **argv)
{
   int rc;

   /* call of INFLOOP with argument PARADOX */
   rc = spawnl(P_WAIT, INFLOOP, INFLOOP, PARADOX, NULL);
   /* Exit on error! */
   assert(errno == EZERO);

   if (!rc)
   {
      /* Here is rc == 0 (that means (INFLOOP PARADOX == 0)).       */
      /* Hence the program INFLOOP has taken the decision, that the */
      /* program PARADOX DOES NEVER encounter any infinite loop and */
      /* that the program PARADOX terminates correctly.             */
      /* But the next step in PARADOX is an infinite loop.          */
      /* With that is shown:                                        */
      /* (INFLOOP PARADOX == 0)  =>  (INFLOOP PARADOX != 0)         */
      printf("Infinite loop in \"%s\"!\n", PARADOX);
      while (TRUE)
      {
      }
   }
   else
   {
      /* Here is rc != 0 (that means (INFLOOP PARADOX != 0)).       */
      /* Hence the program INFLOOP has taken the decision, that the */
      /* program PARADOX DOES ALWAYS encounter an infinite loop.    */
      /* But here PARADOX terminates correctly WITHOUT encountering */
      /* any infinite loop.                                         */
      /* With that is shown:                                        */
      /* (INFLOOP PARADOX != 0)  =>  (INFLOOP PARADOX == 0)         */
      printf("\"%s\" has terminated correctly ", PARADOX);
      printf("WITHOUT encountering any infinite loop.\n");
   }

   return(0);
}
```

‚Paradox.c'

2. Machine Model

We restrict ourselves now on a simple as possible machine model. A computer is a quadruple (m, M, S, R) composed out of a number $m \in \mathbb{N}_+$, the finite, non-empty set $M := \{0, \ldots, 255\}^{\{1,\ldots,m\}}$ (memory), a function $S : M \to M$ (step-function) and a function $R : M \to \{0, 1\}$ (running function).

This simple modell would be possible as we theoretically can integrate the different types of memory

> accumulator, flags, registers, instruction pointer, stack, stack pointer, files, RAM und ROM

into M.

The step-function S simulates principally the processor of a real computer in single step mode. Since S is a function in a mathematical sense, S is as a relation left total and single valued. Left total means, that with **every** state $z \in M$ a further state $\tilde{z} \in M$ exists, so that $(z, \tilde{z}) \in S$ (i. e. the state $\tilde{z}$ is generated by S from state z). Single-valued means, that the machine is **deterministic**.

Because of $\# M < \infty$, the machine is **finite**.

A Program P on the machine (m, M, S, R) is per definitionem a sequence $(P_i)_{i \in \mathbb{N}_0} \in M^{\mathbb{N}_0}$ in M with $\forall i \in \mathbb{N}_0 \quad P_{i+1} = S(P_i)$.

Then the running-function is described by the following statement:

For all programs P on the maschine (m, M, S, R) it holds

$$(P \text{ terminates correctly}) \quad \Leftrightarrow \quad \left(\exists i \in \mathbb{N}_0 \quad R\left(P_i\right) = 0\right)$$

So we can define for all programs P on the machine (m, M, S, R), which terminates correctly, the result ρ of the program P through

$$\rho = P_{\min\left\{i \in \mathbb{N}_0 : \; R\left(P_i\right)=0\right\}}$$

Be Ω the class of all such machines.

3. The Way Out of The Halting-Problem

Lemma:

Pre.: Let ω be a machine of the class Ω.

Ass.: Every program P on ω terminates correctly or ends up in an infinite loop.

3.1. Theorem

Ass.: For every machine $\omega = (m, M, S, R)$ of class Ω it is possible to construct a machine $\hat{\omega} = (\hat{m}, \hat{M}, \hat{S}, \hat{R})$ of class Ω in such a way that the following is true:

1. $m < m + m(\# M + 1) \leq \hat{m}$

2. There exists a programm Q on $\hat{\omega}$ with the following properties:

   1. Q terminates correctly

   2. For every program P on ω exists one and only one program $\hat{P}$ on $\hat{\omega}$ with the following property:

$$\forall i \in \{1, \ldots, \hat{m}\} \quad (\hat{P}_0)(i) = \begin{cases} (P_0)(i) & i \leq \mathrm{m} \\ (Q_0)(i) & i > m \end{cases}$$

   3. For every program P on ω it holds:

      $\hat{P}$ terminates correctly

   4. For every program P on ω it is possible to decide by the result of the program $\hat{P}$ wether P terminates correctly or ends up in an infinite loop.

Proof: constructive sketch (see custos.zip)

3.2. What's about 1.?

First one must realize, that the source-code in 1.3. strictly speaking is not a program, but a program-template. This program-template has to be compiled with a program on a machine as the result. These resulting programs inter alia depend on the compiler and the machine, so they can be different.

Then it is clear:

The way out of the paradoxon in 1. is possible, because the "monitoring" program Q on $\hat{\omega}$ from 3.1. only can "monitor" all programs on ω and not programs on $\hat{\omega}$.

4. Time

For a real machine of the class Ω one can decide by the amount of time a program needs for running, wether the program has ended up in an infinite loop.

Let $\omega = (m, M, S, R)$ be a machine of the class Ω. We now define a mapping $T : M \to \mathbb{R}_+$ through

$$\forall z \in M \quad \begin{pmatrix} T(z) \text{ is the time, which the machine } \omega \\ \text{needs, to compute } S(z). \end{pmatrix}$$

At last we define $\tau \in \mathbb{R}_+$ through

$$\tau := \# M \left(\max \left(\left\{ t \in \mathbb{R}_+ : \quad \exists z \in M \quad t = T(z) \right\} \right) \right)$$

Then for every program P on ω it is valid:

$$\begin{pmatrix} \text{If } P \text{ runs longer than } \tau, \text{ then } P \\ \text{has ended up in an infinite loop.} \end{pmatrix} \qquad (*)$$

Proof:

We have to show:

$$\forall \left(P \text{ program on } \omega\right) \quad \begin{pmatrix}\text{If the program } P \text{ terminates correctly,}\\ \text{then } P \text{ runs at most until } \tau.\end{pmatrix}$$

Let $P = \left(P_i\right)_{i \in \mathbb{N}_0} \in M^{\mathbb{N}_0}$ be a program on ω, which terminates correctly. Because P terminates correctly, there exists $i_0 \in \mathbb{N}_0$ with

$$i_0 := \min\left\{i \in \mathbb{N}_0 : \quad R\left(P_i\right) = 0\right\} \tag{1}$$

without loss of generality $i_0 \geq 1$

$$\begin{pmatrix}\text{With } i_0 = 0 \text{ it follows:}\\ \text{The running time of } P \text{ is } 0\end{pmatrix}$$

We now define:

$$I_0 := \left\{j \in \mathbb{N}_0 : \quad j < i_0\right\} \neq \emptyset \tag{2}$$

Then we have:

$$\forall j \in I_0 \quad R\left(P_j\right) \neq 0 \tag{3}$$

Then for the running time $t_P \in \mathbb{R}_+$ of P the following is valid:

$$t_P \leq \sum_{j \in I_0} T\left(P_j\right) \tag{4}$$

respectively

$$t_P \leq \left(\# I_0\right) \max\left(\left\{t \in \mathbb{R}_+ : \quad \exists z \in M \quad t = T\left(z\right)\right\}\right) \tag{5}$$

We now define a mapping $f: I_0 \to M$ through

$$\forall j \in I_0 \quad f(j) \coloneqq P_j \tag{6}$$

Because of (5) and (6) we have to show:

$$f: I_0 \to M \text{ is injective} \tag{7}$$

Proof of (7):

Supp.: $f: I_0 \to M$ is not injective

Because of (1), (2) and (3) there exists $k, l \in I_0$ with

$$k < l \quad \text{and} \quad f(k) = f(l) \tag{6}$$

Because P is a program on ω, we have:

$$\forall j \in I_0 \quad f(j) = P_j = S^j(P_0) \tag{7}$$

With (6) and (7) follows:

$$S^k(P_0) = S^l(P_0) \tag{8}$$

With this (Cave! (2)) we have also:

$$S^{i_0 - l}\left(S^k(P_0)\right) = S^{i_0 - l}\left(S^l(P_0)\right) \tag{9}$$

respectively

$$P_{i_0 - l + k} = S^{i_0 - l + k}(P_0) = S^{i_0}(P_0) = P_{i_0} \tag{10}$$

With $(i_0 - l + k) \in I_0$, (1) and (3) the following is true:

$$0 \neq R\left(P_{i_0 - l + k}\right) = R\left(P_{i_0}\right) = 0 \tag{11}$$

This is a contradiction!

4. Index of Literature

[1] Lectures in theoretical Informatics 1993 - 1997
University of Cologne

[2] www.Wikipedia.org
Internet