1. Einführung

1.1. Turing-Maschinen und das Halteproblem nach [1]

Satz: Das Halteproblem ist für Turing-Maschinen unentscheidbar.

1.2. Illustration nach [2]

Die Frage ist im Prinzip, ob es möglich ist, auf einer gegebenen Maschine *M* ein Programm zu schreiben, das **alle** Programme auf *M* dahingehend beurteilt, ob sie in eine Endlosschleife geraten oder korrekt terminieren.

Ein diesbezügliches Gedankenexperiment führt auf ein Paradoxon:

Angenommen, auf einem UNIX-Rechner *U* existiert ein Programm ‚*InfLoop*' mit den Eigenschaften

> *InfLoop* wird ein Kommandozeilenargument übergeben

und

> Für alle Programme *P* auf *U*, die keine Kommandozeilenargumente akzeptieren, gilt:
>
> 1. (*InfLoop P*) terminiert korrekt
> 2. (*InfLoop P* = 0) ⇔ (*P* terminiert korrekt)

Dann kann man ein Programm ‚*Paradox*' auf *U* konstruieren, das die Existenz von ‚*InfLoop*' zu einem Widerspruch führt (C-Code von ‚Paradox' siehe nächste Seite 1.3.). Es gilt nämlich mit diesem Programm ‚*Paradox*':

(*InfLoop Paradox* = 0) ⇔ (*InfLoop Paradox* ≠ 0)

Widerspruch!

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. Maschinenmodell

Wir beschränken uns hier auf ein möglichst einfaches Maschinenmodell. Ein Rechner ist ein Quadrupel (m, M, S, R), das aus einer Zahl $m \in \mathbb{N}_+$, der endlichen, nicht-leeren Menge $M := \{0, \ldots, 255\}^{\{1,\ldots,m\}}$ (dem Memory), einer Funktion $S : M \to M$ (der Step-Funktion) und einer Funktion $R : M \to \{0, 1\}$ (der Running-Funktion) besteht.

Dieses einfache Modell wird dadurch möglich, daß man die Speicherarten

> Akkumulator, Flags, Register, Instruction Pointer, Stack, Stack Pointer, Dateien, RAM und ROM

theoretisch unter M vereinen kann.

Die Step-Funktion S simuliert im Prinzip den Prozessor eines realen Rechners im Einzelschrittverfahren. Da S eine Funktion im mathematischen Sinne ist, ist S als Relation linkstotal und rechtseindeutig. Linkstotal bedeutet, daß zu **jedem** Zustand $z \in M$ ein weiterer Zustand $\tilde{z} \in M$ existiert, so daß $(z, \tilde{z}) \in S$. (d. h. der Zustand $\tilde{z}$ geht mittels S aus dem Zustand z hervor). Rechtseindeutig bedeutet, daß die Maschine **deterministisch** ist.

Wegen $\# M < \infty$ ist die Maschine **endlich**.

Ein Programm P auf der Maschine (m, M, S, R) ist eine Folge $(P_i)_{i \in \mathbb{N}_0} \in M^{\mathbb{N}_0}$ in M mit $\forall i \in \mathbb{N}_0 \quad P_{i+1} = S(P_i)$.

Die Running-Funktion wird dann durch folgende Aussage charakterisiert:

Für alle Programme P auf der Maschine (m, M, S, R) gilt:

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

Dann kann man für alle Programme P auf der Maschine (m, M, S, R), die korrekt terminieren, das Resultat ρ des Programms P durch folgende Aussage definieren

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

Bezeichne Ω die Klasse aller solchen Maschinen.

3. Ausweg aus dem Halteproblem

Lemma:

Vor.: Sei ω eine Maschine der Klasse Ω.

Beh.: Jedes Programm P auf ω terminiert korrekt oder gerät in eine Endlosschleife.

3.1. Theorem

Beh.: Zu jeder Maschine $\omega = (m, M, S, R)$ der Klasse Ω kann man eine Maschine $\hat{\omega} = (\hat{m}, \hat{M}, \hat{S}, \hat{R})$ der Klasse Ω konstruieren, so daß gilt:

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

2. Es existiert ein Programm Q auf $\hat{\omega}$ mit den Eigenschaften:

   1. Q terminiert korrekt

   2. Zu jedem Programm P auf ω existiert genau ein Programm $\hat{P}$ auf $\hat{\omega}$ mit folgender Eigenschaft:

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

   3. Für jedes Program P auf ω gilt:

      $\hat{P}$ terminiert korrekt

   4. Für jedes Program P auf ω kann man am Resultat von $\hat{P}$ ablesen, ob P korrekt terminiert oder in eine Endlosschleife gerät.

Bew.: durch konstruktive Beweisskizze (siehe custos.zip)

3.2. Was ist mit 1.?

Zunächst ist zu bemerken, daß der Quelltext in 1.3. streng genommen kein Programm ist, sondern eine Programm-Schablone, aus der mittels eines Compilers ein Programm auf einer gegebenen Maschine wird. Die so entstandenen Programme hängen unter anderem vom Compiler und von der Maschine ab, so daß sie unterschiedlich sein können.

Dann wird klar:

Der Ausweg aus dem Paradoxon aus 1. ist möglich, weil das „überwachende" Programm Q auf $\hat{\omega}$ aus 3.1. nur alle Programme auf ω „überwachen" kann und nicht Programme auf $\hat{\omega}$.

4. Zeit

Bei einer realen Maschine der Klasse Ω kann man am Zeitverbrauch eines Programms abmessen, ob das Programm in eine Endlosschleife geraten ist.

Sei $\omega = (m, M, S, R)$ eine Maschine der Klasse Ω. Wir definieren dann eine Abbildung $T : M \to \mathbb{R}_+$ durch

$$\forall z \in M \quad \begin{pmatrix} T(z) \text{ ist die Zeit, die die Maschine } \omega \\ \text{benötigt, um } S(z) \text{ zu berechnen.} \end{pmatrix}$$

Außerdem definieren wir $\tau \in \mathbb{R}_+$ durch

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

Dann gilt für jedes Programm P auf ω:

$$\begin{pmatrix} \text{Dauert der Lauf des Programms } P \text{ länger als } \tau, \\ \text{so ist es in eine Endlosschleife geraten.} \end{pmatrix} \qquad (*)$$

Beweis hiervon:

Zu zeigen ist:

$$\forall \left(P \text{ Programm auf } \omega\right) \quad \begin{pmatrix} \text{Terminiert das Programm } P \\ \text{korrekt, so dauert der} \\ \text{Lauf von } P \text{ höchstens } \tau. \end{pmatrix}$$

Sei $P = \left(P_i\right)_{i \in \mathbb{N}_0} \in M^{\mathbb{N}_0}$ ein Programm auf ω, das korrekt terminiert. Da P korrekt terminiert, existiert $i_0 \in \mathbb{N}_0$ mit

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

o.B.d.A gelte $i_0 \geq 1$, denn

$$\begin{pmatrix} \text{Aus } i_0 = 0 \text{ folgt:} \\ \text{Die Laufzeit von } P \text{ ist } 0 \end{pmatrix}$$

Wir setzen dann:

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

Damit gilt:

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

Dann gilt für die Laufzeit $t_P \in \mathbb{R}_+$ von P:

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

bzw.

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

Wir definieren nun noch eine Abbildung $f: I_0 \to M$ durch

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

Wegen (5) und (6) bleibt nur noch zu zeigen:

$$f: I_0 \to M \text{ ist injektiv} \tag{7}$$

Beweis von (7):

Ann.: $f: I_0 \to M$ ist nicht injektiv

Wegen (1), (2) und (3) existieren $k, l \in I_0$ mit

$$k < l \quad \text{und} \quad f(k) = f(l) \tag{8}$$

Da P ein Programm auf ω ist, gilt dabei:

$$\forall j \in I_0 \quad f(j) = P_j = S^j(P_0) \tag{9}$$

Aus (8) und (9) folgt:

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

Damit hat man aber auch (Cave! (2)):

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

bzw.

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

Wegen $(i_0 - l + k) \in I_0$, (1) und (3) gilt dann:

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

Dies ist ein Widerspruch!

5. Literaturverzeichnis

[1] Vorlesungen über theoretische Informatik 1993 – 1997
Universität zu Köln

[2] www.Wikipedia.org
Internet