# Halteproblem und Church'sche These

## Aufgabe 1 ( Halteproblem )

Das Halteproblem (Alan Turing) lautet:

> *Es gibt kein Programm, welches feststellt, ob ein beliebiges Programm ein Ergebnis erzielt oder unendlich lange ausgeführt wird.*

Erklären Sie anhand des folgenden Beispiels, warum das Halteproblem nicht lösbar ist.

Sei p ein Programm und x eine Zeichenfolge als Eingabe für p.

**Frage:** Gibt es eine *boolesche Funktion*, die entscheidet, ob p angewandt auf x hält?

Bei positiver Antwort hätte man ein sehr mächtiges Programm, welches entscheidet, ob ein beliebiges Programm p bei Eingabe einer beliebigen Zeichenkette x hält oder nicht hält.

Da das Programm p selbst ein Text ist, können wir die gesuchte boolesche Funktion wie folgt schreiben.

```
function haelt (p, x : TEXT) : boolean;
begin
  if  <p terminiert bei x>
    then    haelt := TRUE
    else    haelt := FALSE
end;
```

Untersuchen Sie nun folgendes Programm *seltsam*, wenn für x das Programm p = *seltsam* selbst eingegeben wird. Erklären Sie den Widerspruch, der durch diese durchaus legitime Eingabe erzeugt wird.

```
program seltsam;
function haelt …;
begin
  lies(p);
  while  haelt(p, p) do;
  writeln(‚fertig')
end.
```

## Aufgabe 2 ( Church'sche These )

Erklären und diskutieren Sie die Church'sche These.

**Aufgabe 3** ( asymptoische Laufzeitanalysen )

Betrachten Sie die drei Algorithmen *Methode-1*, *Methode-2* und *Methode-3* für das Problem c ∈ S. Welche asymptotische Laufzeit haben diese drei Algorithmen? Analysieren Sie den Pseudocode und schreiben Sie die Laufzeit in O-Notation auf.

**Algorithmus 1:**

```
Output:  true, falls c ∈ S, sonst false
Methode-1: contains
    var b : bool;
    b := false;
    for  i := 1 to n do
        if  S[i] = c  then  b := true  endif;
    endfor;
    return  b;
    end  contains.
```

**Algorithmus 2:**

```
Output:  true, falls c ∈ S, sonst false
Methode-2: contains
i := 1;
while S[i]  ≠ c  and  i ≤ n  do  i := i+1;  { stop, wenn c gefunden }
if  i ≤ n     then return true
                 else return  false
```

**Algorithmus 3:**

**Input:** S: aufsteigend sortiertes Array von Integerzahlen
low, high: unterer und oberer Indes des zu Bereichs von S
c: integer

**Output:** true: falls c im Bereich S[low] ..S[high] vorkommt
false: sonst

**Methode 3:** binäre Suche

**Aufgabe 4**

Warum gilt folgendes? $\log_3 n = O(\log n)$

$$\log_{10} n = O(\log n)$$

**Aufgabe 5** ( Logarithmieren )

Gegeben ist die Polynomfunktion $y = bx^c$

Welches Ergebnis erhält man durch Logarithmieren dieser Funktion?

**Lösung Aufgabe 1**

```
function haelt (p, x : TEXT) : boolean;
begin
  if  <p terminiert bei x>
    then    haelt := TRUE
    else    haelt := FALSE
end;


program seltsam;
function haelt …;
begin
  lies(p);
  while  haelt(p, p) do;
  writeln(,fertig')
end.
```

**Fall 1:** *seltsam* stoppt, somit wird *haelt* = TRUE, also kommt *seltsam* in eine Endlosschleife, womit *seltsam* nicht stoppt!

**Fall 2:** *seltsam* stoppt nicht, somit wird *haelt* = FALSE, also schreibt *seltsam* ‚fertig', womit *seltsam* stoppt!

In beiden Fällen entsteht ein Widerspruch: Das Programm *seltsam* stoppt genau dann, wenn es nicht stoppt.

**Lösung Aufgabe 2** ( → Diskussion der Church'schen These )

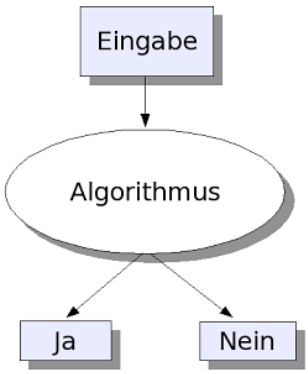
**Churchsche These (auch Church-Turing-These)**: Jede im intuitiven Sinne berechenbare Funktion, d.h. eine Funktion, die prinzipiell auch von einem Menschen ausgerechnet werden könnte, ist auch Turing-berechenbar.

**Intuitiv berechenbar.** Diese These ist wohl nicht beweisbar, da der Begriff *intuitiv berechenbare Funktion* nicht exakt formalisiert werden kann. Man versteht darunter alle Funktionen, die prinzipiell auch von einem Menschen *ausgerechnet* werden könnten.
Das Problem liegt vor allem darin, dass das, was der Mensch kann, nicht formal beschreibbar ist. Wäre es möglich, das menschliche Gedächtnis formal zu beschreiben, dann könnte man auch intelligente Computer bauen.

**Turing-Maschine**. Modell für die Klasse der berechenbaren Funktionen. Alan Turing beabsichtigte, mit der Turingmaschine ein Modell des mathematisch arbeitenden Menschen zu schaffen und damit eine mathematische Definition des Begriffs „Algorithmus" zu formulieren.

**Churchsche These.** Die Klasse der Turing-berechenbaren Funktionen ist genau die Klasse der intuitiv berechenbaren Funktionen.
In der mathematischen Logik gibt es verschiedene Ansätze zur formalen Definition eines Algorithmus. Ein Ansatz ist z.B. der Entwurf der Turing-Maschine (A.M. Turing 1936) oder das **Lambda-Kalkül** von A. Church und Stephen Kleene 1936. Church erkannte, dass man zu jedem, in irgendeiner Form definierten Algorithmus eine zugehörige Turing-Maschine konstruieren kann. Er formulierte daher die Churchsche These.

Der **Lambda-Kalkül** ist eine **formale Sprache** zur Untersuchung von Funktionen, die Funktionsdefinitionen, das Definieren formaler, sowie das Auswerten und Einsetzen aktueller Parameter regelt. Church benutzte ihn, um 1936 eine negative Antwort auf das **Entscheidungsproblem** zu geben. Eine Eigenschaft auf einer Menge heißt **entscheidbar** (auch: rekursiv), wenn es ein Entscheidungsverfahren für sie gibt. Ein **Entscheidungsverfahren** ist ein **Algorithmus**, der für jedes Element der Menge beantworten kann, ob es die Eigenschaft hat oder nicht. Wenn es ein solches Entscheidungsverfahren nicht gibt, dann nennt man die Eigenschaft **unentscheidbar**. Als **Entscheidungsproblem** bezeichnet man die Frage, ob und wie für eine gegebene Eigenschaft ein Entscheidungsverfahren formuliert werden kann. Struktur eines Entscheidungsproblems:




Eine Teilmenge T einer Menge M heißt **entscheidbar**, wenn ihre charakteristische Funktion **berechenbar** ist. Der Entscheidbarkeitsbegriff ist somit auf den Berechenbarkeitsbegriff zurückgeführt.

Ganz **einfach ausgedrückt** sagt die Church'sche These aus, dass Funktionen, deren Berechenbarkeit anhand von diversen mathematischen Modellen gezeigt werden kann, auch maschinell berechenbar sind.

Da man das, was der Mensch kann, nicht formal beschreiben kann, ist der Beweis der Church'schen These auch so problematisch und bis heute nicht gelungen. Fraglich ist, ob er überhaupt jemals gelingt.

**Falls die These wahr ist**, kann es kein Rechenmodell geben, das mehr als die bisherigen Modelle berechnen kann. Insbesondere ist ein Computer ein solches Rechenmodell, somit kann auf ihm theoretisch jeder Algorithmus ausgeführt werden, vorausgesetzt genügend Speicherplatz ist vorhanden. Es ist dann nicht möglich eine Rechenmaschine zu bauen, die mehr berechnen kann als ein Computer bereits kann. Da viele Programmiersprachen ebenfalls turing-vollständig sind, kann man jeglichen Algorithmus mittels eines Quelltexts dieser Sprachen ausdrücken. Insbesondere können sich verschiedene Rechenmodelle gegenseitig simulieren.

**Lösung Aufgabe 3**

**Analyse Methode-1:**

$T_{best}$ = O(n)
$T_{worst}$ = O(n)
$T_{avg}$ = O(n)

**Analyse Methode-2:**

$T_{best}$ = O(1)
$T_{worst}$ = O(n)
$T_{avg}$ = ?

**Fall 1**: c kommt unter den n Elementen in S vor und zwar mit gleicher Wahrscheinlichkeit auf Platz 1, Platz 2, ..., Platz n. Also gilt:

$$T_1 = \frac{1}{n} \cdot 1 + \frac{1}{n} \cdot 2 + \cdots + \frac{1}{n} \cdot n = \frac{1}{n} \cdot \sum_{i=1}^{n} i = \frac{1}{n} \cdot \frac{n \cdot (n-1)}{2} = \frac{n+1}{2}$$

**Fall 2**: c kommt in S **nicht** vor: $T_2(n) = n$

Da beide Fälle nach der obigen Annahme je mit einer Wahrscheinlichkeit von 0,5 vorkommen sollen, gilt:

$$T_{avg} = \frac{1}{2} \cdot T_1(n) + \frac{1}{2} T_2(n) = \frac{1}{2} \cdot \frac{n}{2} + \frac{n}{2} = \frac{3}{4} \cdot n$$

**Analyse Methode-3:**

$T(0) = a,\ T(m) = b + T(m/2)$

$T(m) = b + T(m/2)$
$= b + b + T(m/4)$
$= b + b + b + T(m/8)$
$\ldots$
$= b + b + b + b + \ldots + b + a$
so oft m halbiert werden kann, bis man 1 erhält
$= b * \log m + a$
$= O(\log m) = T_{worst}$

Also folgt: $T_{worst}(n) = O(\log n)$
$T_{best}(n) = O(1)$, wenn c = S[(n+1)/2]

## Lösung Aufgabe 4

Warum gilt folgendes? $log_3 n = O(\log n)$

$$log_{10} n = O(\log n)$$

Antwort: Verschiedene Logarithmen unterscheiden sich nur durch einen konstanten Faktor. Zum Beispiel: $\log_a n$ und $\log_b n$ kann man sehr einfach ineinander überführen. Den Wert $\log_a n$ erhält man aus $\log_b n$ indem man den Wert $\log_b n$ mit $1/\log_b a$ multipliziert.

$$log_a n = \frac{log_b n}{log_b a}$$

## Lösung Aufgabe 5

Gegeben ist die Polynomfunktion $y = bx^c$

Welches Ergebnis erhält man beim Logarithmieren?

Setze $y' = log_{10} bx^c = log_{10} x^c + log_{10} b = c \cdot log_{10} x + log_{10} b$

Setze $x' = log_{10} x$

Mit x' und y' erhält man eine Gerade: $y' = cx' + b$

Beispiel: $y = 2x^3$

dann: $y' = 3x' + log_{10} 2 = 3x' + 0{,}30103$

Durch Potenzierung erhält man das Polynom $y = 2x^3$ zurück.

b: Schnittpunkt der Geraden mit der y-Achse
c: Steigung der Geraden

Beispiel: $t'(n) = \frac{4}{3} n + 2$

Delogaritmieren: $t(n) = 10^2 \cdot n^{4/3}$

| b | c |
|---|---|
| 1 | 4 |

$$y = b \cdot x^{c}$$

| x | y | x'=log₁₀(x) | y'=log₁₀(y) |
|---|---|---|---|
| | | | |
| 1 | 1,00 | 0,00000 | 0,00000 |
| 2 | 16,00 | 0,30103 | 1,20412 |
| 3 | 81,00 | 0,47712 | 1,90849 |
| 4 | 256,00 | 0,60206 | 2,40824 |
| 5 | 625,00 | 0,69897 | 2,79588 |
| 6 | 1296,00 | 0,77815 | 3,11261 |
| 7 | 2401,00 | 0,84510 | 3,38039 |
| 8 | 4096,00 | 0,90309 | 3,61236 |
| 9 | 6561,00 | 0,95424 | 3,81697 |
| 10 | 10000,00 | 1,00000 | 4,00000 |