

Sachen gibt's, die gibt's gar nicht

Computer können doch nicht alles

Rolf Wanka

Department Informatik

Universität Erlangen-Nürnberg



Überblick

1. Wie berechnet man $f(n) = n^2$?
2. Warum kann man Endlosschleifen nicht vorab erkennen?
3. Warum kann man nicht alle Quadrierer vorab erkennen?



Der erste Quadrierer

Das folgende Programm berechnet die Funktion $f(n) = n^2$:

```
lies  $n$ ;  
gib  $n * n$  aus;
```

Der zweite Quadrierer

Auch dieses Programm berechnet die Funktion $f(n) = n^2$:

```
lies  $n$ ;  
 $q := 0$ ;  
for  $i := 1$  to  $n$  do  
     $q := q + n$  ;  
gib  $q$  aus;
```

Es realisiert $n^2 = \underbrace{n + \dots + n}_{n \text{ Mal}} = \sum_{i=1}^n n$.

Der dritte Quadrierer

Dieses Programm berechnet ebenfalls die Funktion $f(n) = n^2$:

```
lies  $n$ ;  
 $q := 0$ ;  
for  $i := 1$  to  $n$  do  
    for  $j := 1$  to  $n$  do  
         $q := q + 1$  ;  
gib  $q$  aus;
```

Es realisiert $n^2 = \sum_{i=1}^n \sum_{j=1}^n 1$.

???

Und was macht dieses Programm?

```
lies  $n$ ;  
 $q := 0$ ;  
for  $i := 1$  to  $n$  do  
     $q := q + (2 * i - 1)$  ;  
gib  $q$  aus;
```

Es realisiert $1 + 3 + 5 + \dots + (2n - 1) = \sum_{i=1}^n (2i - 1)$.

Der vierte Quadrierer

```
lies n;  
q := 0;  
for i := 1 to n do  
    q := q + (2 * i - 1) ;  
gib q aus;
```

berechnet wieder die Funktion $f(n) = n^2$.

Weil $i^2 - (i - 1)^2 = i^2 - (i^2 - 2i + 1) = 2i - 1$ gilt,
ist $\sum_{i=1}^n (2i - 1) = \sum_{i=1}^n (i^2 - (i - 1)^2) = n^2$.

Der vierte Quadrierer

```
lies  $n$ ;  
 $q := 0$ ;  
for  $i := 1$  to  $n$  do  
     $q := q + (2 * i - 1)$  ;  
gib  $q$  aus;
```

berechnet wieder die Funktion $f(n) = n^2$.

Weil $i^2 - (i - 1)^2 = i^2 - (i^2 - 2i + 1) = 2i - 1$ gilt,
ist $\sum_{i=1}^n (2i - 1) = \sum_{i=1}^n (i^2 - (i - 1)^2) = n^2$.

Der fünfte Quadrierer

Dieses Programm berechnet schon wieder $f(n) = n^2$:

```
lies  $n$ ;  
 $q := -n$ ;  
for  $i := 1$  to  $n$  do  
     $q := q + 2 * i$  ;  
gib  $q$  aus;
```

Der sechste (und letzte) Quadrierer

Und unser (für heute) letzter Quadrierer:

lies n ;

$q := 17 - 4 * n + 22$; (* eigentlich bedeutungslos 😊 *)

$q := -n$;

for $i := 1$ **to** n **do**

$q := q + 2 * i$;

gib q aus;

Wieviele verschiedene Quadrierer gibt es?

Es gibt offenbar sehr, sehr viele verschiedene Programme, die Quadrierer sind.

Um genau zu sein: unendlich viele!

Ein Quadrierer-Entscheider

Wenn es nun so viele verschiedene Programme gibt, die Quadrierer sind, wäre es gut, eine Bibliotheksfunktion

function Ist_Quadrierer (P : Programmtext) : {**ja**,**nein**}

zu haben mit:

Programm P ist ein Quadrierer \Rightarrow Ist_Quadrierer(P) liefert **ja**

Programm P ist **kein** Quadrierer \Rightarrow Ist_Quadrierer(P) liefert **nein**

Man sagt: Ist_Quadrierer(P) **entscheidet**, ob P ein Quadrierer ist.

Das Problem der Endlosschleifen

Quadrierer zu entscheiden, ist (im Moment jedenfalls) vielleicht nicht sooo interessant.

Wie ist es mit Endlosschleifen?

```
program Gruß ;  
  lies n ;  
666: if n gerade then goto 666 ;  
  gib „Hallo Welt“ aus;
```

Nun gilt:

Gruß gestartet mit 5 hält.

Gruß gestartet mit 4 hält nicht.

Das Problem der Endlosschleifen

Es wäre unglaublich nützlich, folgende Bibliotheksfunktion

function HÄLT (P: Programmtext, n: Eingabetext): {**ja,nein**}

zu haben mit:

Programm P gestartet mit n hält \Rightarrow HÄLT(P,n) liefert **ja**

Programm P gestartet mit n hält **nicht** \Rightarrow HÄLT(P,n) liefert **nein**

Man sagt: HÄLT **entscheidet** das **Halteproblem**.

Das Problem der Endlosschleifen

In unserem Beispiel wäre:

HÄLT(Gruß,4) = **nein**

und

HÄLT(Gruß,5) = **ja**

Das Problem der Endlosschleifen

Warum ist in keiner Programmbibliothek eine derart nützliche Bibliotheksfunktion enthalten?



Das Problem der Endlosschleifen

Warum ist in keiner Programmbibliothek eine derart nützliche Bibliotheksfunktion enthalten?

Das liegt daran, dass es HÄLT nicht gibt!

Das Problem der Endlosschleifen

Warum ist in keiner Programmbibliothek eine derart nützliche Bibliotheksfunktion enthalten?

Das liegt daran, dass es HÄLT nicht gibt!

Wir nehmen einmal an, dass irgendjemand

```
function HÄLT ( P: Programmtext, n: Eingabe ): {ja,nein}  
begin ... end;
```

geschrieben hat, **es also HÄLT gibt**.

Wir schreiben ein weiteres Programm!

Das Problem der Endlosschleifen

- (1) **program** TRICK (input, output);
- (2) **var** P : Text ;
- (3) **function** HÄLT (P,n:Text): boolean ;
- (4) **begin ... end**;
- (5) **begin**
- (6) lies(P) ;
- (7) **while** HÄLT(P,P) **do** writeln('Ich mache weiter') ;
- (8) writeln('Ich höre jetzt auf')
- (9) **end.**

Das Problem der Endlosschleifen

TRICK :=

```
“program TRICK (input, output);  
var P : Text ;  
function HÄLT (P, n:Text): boolean ;  
begin ... end;  
begin  
  lies(P) ;  
  while HÄLT(P,P) do writeln('Ich mache weiter') ;  
  writeln('Ich höre jetzt auf')  
end.”
```

TRICK ist der bloße Programm-Text.

Das Problem der Endlosschleifen

Eine legitime Frage ist nun:

Hält **TRICK** gestartet mit *TRICK* ?

Das Problem der Endlosschleifen

Eine legitime Frage ist nun:

Hält **TRICK** gestartet mit *TRICK* ?

Falls ja:

TRICK gestartet mit *TRICK* gibt „Ich höre jetzt auf“ aus;

Das Problem der Endlosschleifen

Eine legitime Frage ist nun:

Hält **TRICK** gestartet mit *TRICK* ?

Falls ja:

TRICK gestartet mit *TRICK* gibt „Ich höre jetzt auf“ aus;
Also wurde die **while**-Schleife nicht durchlaufen;

Das Problem der Endlosschleifen

Eine legitime Frage ist nun:

Hält **TRICK** gestartet mit *TRICK* ?

Falls ja:

TRICK gestartet mit *TRICK* gibt „Ich höre jetzt auf“ aus;
Also wurde die **while**-Schleife nicht durchlaufen;
Also gibt **HÄLT**(*TRICK*, *TRICK*) **nein** zurück;

Das Problem der Endlosschleifen

Eine legitime Frage ist nun:

Hält **TRICK** gestartet mit *TRICK* ?

Falls **ja**:

TRICK gestartet mit *TRICK* gibt „Ich höre jetzt auf“ aus;
Also wurde die **while**-Schleife nicht durchlaufen;
Also gibt **HÄLT**(*TRICK*, *TRICK*) **nein** zurück,
was wiederum bedeutet: **TRICK** gestartet mit *TRICK* hält **nicht**.

Das Problem der Endlosschleifen

Eine legitime Frage ist nun:

Hält **TRICK** gestartet mit *TRICK* ?

Falls **ja**:

TRICK gestartet mit *TRICK* gibt „Ich höre jetzt auf“ aus;
Also wurde die **while**-Schleife nicht durchlaufen;
Also gibt **HÄLT**(*TRICK*, *TRICK*) **nein** zurück,
was wiederum bedeutet: **TRICK** gestartet mit *TRICK* hält **nicht**.

Also ist die Annahme „**ja**“ falsch, es muß „**nein**“ heißen ... oder?

Das Problem der Endlosschleifen

Eine legitime Frage ist nun:

Hält **TRICK** gestartet mit *TRICK* ?

Falls **nein**:

TRICK gestartet mit *TRICK* gibt „Ich mache weiter“ aus;

Das Problem der Endlosschleifen

Eine legitime Frage ist nun:

Hält **TRICK** gestartet mit *TRICK* ?

Falls **nein**:

TRICK gestartet mit *TRICK* gibt „Ich mache weiter“ aus;
Also wird die **while**-Schleife durchlaufen;

Das Problem der Endlosschleifen

Eine legitime Frage ist nun:

Hält **TRICK** gestartet mit *TRICK* ?

Falls **nein**:

TRICK gestartet mit *TRICK* gibt „Ich mache weiter“ aus;
Also wird die **while**-Schleife durchlaufen;
Also gibt **HÄLT**(*TRICK*, *TRICK*) **ja** zurück,

Das Problem der Endlosschleifen

Eine legitime Frage ist nun:

Hält **TRICK** gestartet mit *TRICK* ?

Falls **nein**:

TRICK gestartet mit *TRICK* gibt „Ich mache weiter“ aus;
Also wird die **while**-Schleife durchlaufen;
Also gibt **HÄLT**(*TRICK*, *TRICK*) **ja** zurück,
was wiederum bedeutet: **TRICK** gestartet mit *TRICK* hält.

Das Problem der Endlosschleifen

Eine legitime Frage ist nun:

Hält **TRICK** gestartet mit *TRICK* ?

Falls **nein**:

TRICK gestartet mit *TRICK* gibt „Ich mache weiter“ aus;
Also wird die **while**-Schleife durchlaufen;
Also gibt **HÄLT**(*TRICK*, *TRICK*) **ja** zurück,
was wiederum bedeutet: **TRICK** gestartet mit *TRICK* hält.

Also ist die Annahme „**nein**“ falsch, es muß „**ja**“ heißen.

HÄLT gibt es nicht

Da wir aber sonst alles richtig gemacht haben, bleibt nur übrig, dass es die Funktion **HÄLT**(P,n) nicht gibt!

Man kann also **im Allgemeinen** Endlosschleifen nicht im Voraus erkennen.

Man sagt: **Das Halteproblem ist unentscheidbar.**

HÄLT gibt es nicht

Da wir aber sonst alles richtig gemacht haben, bleibt nur übrig, dass es die Funktion **HÄLT**(P,n) nicht gibt!

Man kann also **im Allgemeinen** Endlosschleifen nicht im Voraus erkennen.

Man sagt: **Das Halteproblem ist unentscheidbar.**

Es gibt kein Computer-Programm, das alle möglichen Endlosschleifen, die es gibt, erkennen kann.

Kann man alle Quadrierer erkennen?

Am Anfang hatten wir nach einer Bibliotheksfunktion `Ist_Quadrierer(P)` gefragt, die „sagen“ soll, ob das Programm P ein Quadrierer ist oder nicht.

Gibt es denn wenigstens die?

Kann man alle Quadrierer erkennen?

Am Anfang hatten wir nach einer Bibliotheksfunktion `Ist_Quadrierer(P)` gefragt, die „sagen“ soll, ob das Programm P ein Quadrierer ist oder nicht.

Gibt es denn wenigstens die?

Leider nicht!

Kann man alle Quadrierer erkennen?

Am Anfang hatten wir nach einer Bibliotheksfunktion `Ist_Quadrierer(P)` gefragt, die „sagen“ soll, ob das Programm P ein Quadrierer ist oder nicht.

Gibt es denn wenigstens die?

Leider nicht!

Denn sonst würde es HÄLT doch geben.

Ist_Quadrierer gibt es nicht

Gegeben ist die Frage „Hält P gestartet mit x?“

Wir schreiben dazu folgendes Programm:

```
program der_siebente_Quadrierer?!?
```

```
  lies  $n$ ;
```

```
  starte P mit  $x$  ; (* eigentlich bedeutungslos *)
```

```
   $q := n * n$ ;
```

```
  gib  $q$  aus;
```

Ist_Quadrierer gibt es nicht

program der_siebente_Quadrierer?!?

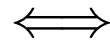
lies n ;

starte P mit x ; (* eigentlich bedeutungslos *)

$q := n * n$;

gib q aus;

der_siebente_Quadrierer?!? ist ein Quadrierer



P gestartet mit der Eingabe x hält.

Ist_Quadrierer gibt es nicht

program der_siebente_Quadrierer?!?

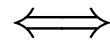
lies n ;

starte P mit x ; (* eigentlich bedeutungslos *)

$q := n * n$;

gib q aus;

der_siebente_Quadrierer?!? ist ein Quadrierer



P gestartet mit der Eingabe x hält.

Würde es `Ist_Quadrierer` geben, würde es also **HÄLT** auch geben.
Also gibt es `Ist_Quadrierer` nicht.

Auf Wiedersehen

Vielen Dank!

Die Folien gibt es zum Download auf

<http://www12.informatik.uni-erlangen.de/people/rwanka>

