

Grundlagen der Theoretischen Informatik

Wintersemester 2024/25

Prof. Dr. Heribert Vollmer

Institut für Theoretische Informatik
Leibniz Universität Hannover

Link für Kurzfragen:



URL: <https://pingo.coactum.de/events/739935>

Inhalt

Sprachen und Grammatiken

Die Chomsky-Hierarchie

Reguläre (Typ-3-) Sprachen

Endliche Automaten

Nichtdeterministische endliche
Automaten

Endliche Automaten und
Typ-3-Grammatiken

Das Pumping Lemma für
reguläre Sprachen

Kontextfreie (Typ-2-) Sprachen

Kellerautomaten

Das Pumping-Lemma für
kontextfreie Sprachen

Typ-1- und Typ-0-Sprachen

Der intuitive Berechenbarkeitsbegriff

Berechenbarkeit durch Maschinen

Turing-Berechenbarkeit

Mehrband-Maschinen

Berechenbarkeit in
Programmiersprachen

Die Programmiersprache LOOP

Die Programmiersprache WHILE

Die Church'sche These

Entscheidbarkeit und Aufzählbarkeit

Unentscheidbare Probleme

Das Halteproblem

Der Satz von Rice

Die Programmiersprache LOOP

Syntaktische Komponenten von LOOP

- ▶ **Variablen:** x_0, x_1, x_2, \dots

Zur besseren Lesbarkeit werden wir auch Variablennamen wie z. B. u, v, x, y, z, \dots benutzen.

- ▶ **Konstanten:** $0, 1, 2, \dots$
- ▶ **Operationszeichen:** + und –
- ▶ **Trennsymbole:** ; und :=
- ▶ **Schlüsselwörter:** LOOP, DO und END

Syntax von LOOP

- Sind x_i und x_j Variablen und c eine Konstante, so sind

$$x_i := x_j + c \quad \text{und} \quad x_i := x_j - c$$

LOOP-Programme.

- Sind P_1 und P_2 LOOP-Programme, so ist

$$P_1; P_2$$

ein LOOP-Programm.

- Ist P ein LOOP-Programm und x_i eine Variable, so ist

$$\text{LOOP } x_i \text{ DO } P \text{ END}$$

ein LOOP-Programm.

Semantik von LOOP

Sei P ein LOOP-Programm. P berechnet eine Funktion
 $f: \mathbb{N}^k \rightarrow \mathbb{N}$ wie folgt:

Zu Beginn der Rechnung befinden sich Eingabewerte
 $n_1, \dots, n_k \in \mathbb{N}$ in den Variablen x_1, \dots, x_k . Alle anderen
Variablen haben den Startwert 0. P wird wie folgt ausgeführt:

- ▶ Durch das Programm „ $x_i := x_j + c$ “ erhält x_i den Wert von $x_j + c$.
- ▶ Durch das Programm „ $x_i := x_j - c$ “ erhält x_i den Wert von $x_j - c$, falls dieser nicht negativ ist, ansonsten den Wert 0.
- ▶ Bei Ausführung von „ $P_1; P_2$ “ wird zunächst P_1 und dann P_2 ausgeführt.
- ▶ Ausführung des Programms „**LOOP** x_i **DO** P' **END**“:
 P' wird so oft ausgeführt, wie der Wert der Variablen x_i zu Beginn angibt, d. h. Zuweisungen an x_i in P' haben keinen Einfluss auf die Anzahl der Wiederholungen.

Ergebnis der Ausführung von P

$f(n_1, \dots, n_k) =$ Wert von x_0 am Ende der Ausführung.

Eine Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$ heißt **LOOP-berechenbar**, falls es ein LOOP-Programm gibt, das f wie soeben festgelegt berechnet.

Beachte: Jedes LOOP-Programm hält nach endlich vielen Schritten an. Daraus folgt, dass jede LOOP-berechenbare Funktion total ist.

Einige spezielle LOOP-Programme

„ $x_i := x_j$ “

steht für

„ $x_i := x_j + 0$ “.

, $x_i := c$ “ (für eine Konstante c)

steht für

, $x_i := x_j + c$ “

(x_j ist eine noch nicht benutzte Variable, die also den Wert 0 hat).

„IF $x_i = 0$ THEN P END“ (für ein LOOP-Programm P)

steht für

„ $x_j := 1;$

LOOP x_i DO $x_j := 0$ END;

LOOP x_j DO P END.“

(x_j ist eine Variable, die in P nicht vorkommt)

„ $x_i := x_j + x_k$ “

steht für

„ $x_i := x_j;$
LOOP x_k DO $x_i := x_i + 1$ END.“

, $x_i := x_j * x_k$ “

steht für

, $x_i := 0;$
LOOP x_k DO $x_i := x_i + x_j$ END.“

Analog:

, $x_i := x_j \text{ DIV } x_k$ "

, $x_i := x_j \text{ MOD } x_k$ "

Die Programmiersprache WHILE

Syntax von WHILE

Erweiterung von LOOP:

neues Schlüsselwort: WHILE

Syntax: Ist P ein WHILE-Programm und x_i eine Variable, so ist

WHILE $x_i \neq 0$ DO P END

ein WHILE-Programm.

Semantik von WHILE

Die Ausführung von „WHILE $x_i \neq 0$ DO P END“ geschieht so, dass Programm P so lange wiederholt ausgeführt wird, wie der Wert von x_i ungleich Null ist.

P berechnet $f: \mathbb{N}^k \rightarrow \mathbb{N}$ wie folgt:

Eingabewerte n_1, \dots, n_k in Variablen x_1, \dots, x_k , die anderen Variablen haben Startwert 0.

$f(n_1, \dots, n_k)$ ist der Wert von x_0 nach der Ausführung von P, falls diese stoppt, ansonsten ist $f(n_1, \dots, n_k)$ undefiniert.

Eine Funktion f heißt **WHILE-berechenbar**, falls es ein WHILE-Programm gibt, das f wie eben festgelegt berechnet.

Vergleich von LOOP und WHILE

Sei $f: \mathbb{N}^k \rightarrow \mathbb{N}$.

f ist LOOP-berechenbar \Leftrightarrow f ist WHILE-berechenbar.

φ ist WHILE-berechenbar:

$\varphi: \mathbb{N} \rightarrow \mathbb{N}$, $\varphi(n) = \text{undef. f.a. } n \in \mathbb{N}$.

$x_0 := 1;$

WHILE $x_0 \neq 0$ DO $x_0 := x_0 + 1$ END;

aber φ ist nicht LOOP-berechenbar.

Beispiel

Das LOOP-Programm

LOOP x DO P END

kann simuliert werden durch

$y := x;$

WHILE $y \neq 0$ DO $y := y - 1$; P END.

(Dabei ist y eine noch nicht verwendete Variable.)

Korollar

Jedes WHILE-Programm ist äquivalent zu (d. h. berechnet die gleiche Funktion) einem WHILE-Programm, in dem keine LOOP-Schleifen vorkommen.

Hauptsatz der Algorithmentheorie

Sei $f: \mathbb{N}^k \rightarrow \mathbb{N}$ für ein $k \geq 1$ (oder: $f: \Sigma^* \rightarrow \Delta^*$)

f ist LOOP-berechenbar.

$\Downarrow \Updownarrow$

f ist WHILE-berechenbar. \iff f ist Turing-berechenbar.

$\Downarrow \Updownarrow$ (Programmiererfahrung)

f ist Python-berechenbar.
Java
C

Algorithmus \equiv TM

Erfahrung:

WHILE-Berechenbarkeit = Java-Berechenbarkeit.

Satz

Jede WHILE-berechenbare Funktion ist Turing-berechenbar.

Satz

Jede Turing-berechenbare Funktion ist WHILE-berechenbar.

Beweis im Skript

Turing \Rightarrow GOTO \Rightarrow WHILE

Beweis des Satzes von F.-97

Sei $f: \mathbb{N}^k \rightarrow \mathbb{N}$, \mathbf{uz}_1 gegeben.

Sei P ein WHILE-Programm, das f berechnet.

Sei x_0, x_1, \dots, x_l alle Variablen, die in P benutzt werden.

O.B.d.A. x_0 nur als Ausgabe.

Idee: Wir verwenden l -Band-TM, die jede Variable auf separatem Band speichert.

Wir simulieren P durch TM M wie folgt:

Induktion:

I.A.: $P \stackrel{\Delta}{=} x_0 := x_0 + c$ wird simuliert durch die TM
Band 0 \mapsto Band 0' $+ c$

I.S.: - $P \stackrel{\Delta}{=} P_1; P_2$. Seien M_1, M_2 TM, die P_1, P_2 simulieren.
(nach I.V.)

Dann wird P simuliert von der TM $M_1; M_2$.

- $P \stackrel{\Delta}{=} \text{WHILE } x_0 \neq 0 \text{ DO } P' \text{ END}$. Sei M' eine TM, die
 P' simuliert (nach I.V.). Dann wird P simuliert
durch die TM WHILE Band $_0 \neq 0$ DO M' END

Eingabe und Ausgabe miteinander. Siehe Skript.

Q

Die Church'sche These

- WHILE-Berechenbarkeit = Java-Berechenbarkeit
 - = C++-Berechenbarkeit
 - = Berechenbarkeit in beliebigen Programmiersprachen
 - = Berechenbarkeit durch Registermaschinen
 - = Berechenbarkeit mit Quanten-Computern
 - = Markov-Berechenbarkeit
 - = λ -Berechenbarkeit
 - = μ -Rekursivitat
 - = Berechenbarkeit in jedem bislang untersuchten formalen System
- WHILE-Berechenbarkeit = **Turing-Berechenbarkeit**

These von Church

Eine Funktion ist berechenbar im intuitiven Sinne, gdw. sie Turing-berechenbar ist.

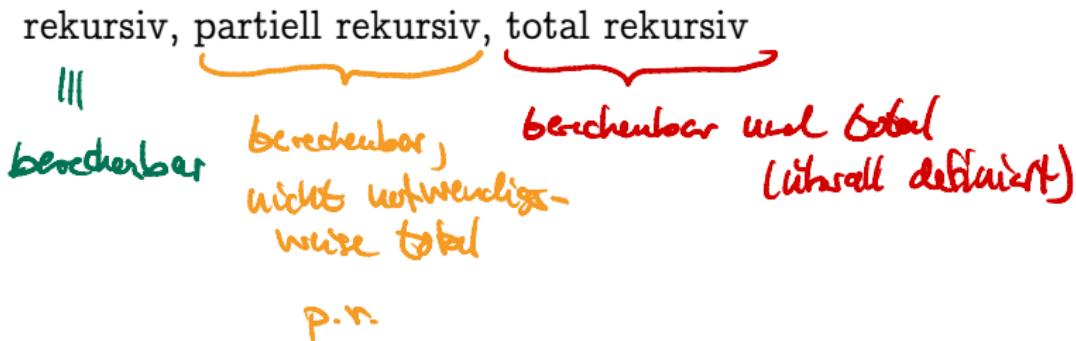
(Nicht beweisbar, da „berechenbar im intuitiven Sinne“ nicht formal gefasst.)

Manchmal auch: „Church-Turing-These“

Allgemeine Sprechweise:

berechenbar \equiv Turing-berechenbar

Weitere gebräuchliche Bezeichnungen:



- ▶ Es gibt WHILE-berechenbare Funktionen, die nicht LOOP-berechenbar sind. 
- ▶ Es gibt totale WHILE-berechenbare Funktionen, die nicht LOOP-berechenbar sind.

Beispiel: Ackermann-Funktion

- ▶ Es gibt WHILE-berechenbare Funktionen, die nicht LOOP-berechenbar sind.
- ▶ Es gibt totale WHILE-berechenbare Funktionen, die nicht LOOP-berechenbar sind.

Beispiel: Ackermann-Funktion