

# Übung 9: Berechnungsmodelle

## Theoretische Informatik Sommersemester 2013

Markus Kaiser

June 25, 2013

## Alle Algorithmen

Typ 0 - Rekursiv aufzählbar  
Turingmaschinen,  $\lambda$ -Kalkül

Typ 1 - Kontextsensitiv  
CSG

Typ 2 - Kontextfrei  
PDA, CFG

Typ 3 - Regulär  
DFA, RE

## Definition (Intuitive Berechenbarkeit)

Eine Funktion  $f : \mathbb{N}^k \mapsto \mathbb{N}$  heißt **intuitiv berechenbar**, wenn es einen Algorithmus gibt, der bei Eingabe  $(n_1, \dots, n_k) \in \mathbb{N}^k$

- nach **endlich vielen Schritten** mit Ergebnis  $f(n_1, \dots, n_k)$  hält, falls  $f(\dots)$  definiert ist,
- und **nicht terminiert**, falls  $f(\dots)$  nicht definiert ist.

## Churchsche These (nicht beweisbar)

Turing-Maschinen können genau **alle** intuitiv berechenbaren Funktionen berechnen.

## Beispiel (Berechenbarkeit)

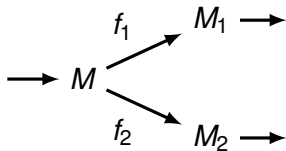
Sind die folgenden Funktionen intuitiv berechenbar?

$$f_1(n) = \begin{cases} 1 & \text{falls } n \text{ prim} \\ 0 & \text{sonst} \end{cases}$$

$$f_2(n) = \begin{cases} 1 & \text{falls } n \text{ die ersten } n \text{ Ziffern von } \pi \text{ darstellt} \\ 0 & \text{sonst} \end{cases}$$

$$f_3(n) = \begin{cases} 1 & \text{falls in } \pi \text{ } n \text{ Nullen am Stück vorkommen} \\ 0 & \text{sonst} \end{cases}$$

Sind  $f_1$  und  $f_2$  Endzustände von  $M$ , so bezeichnet



eine **Fallunterscheidung**.

Beispiel (Band=0?)

$$\delta(q_0, 0) = (q_0, 0, R)$$

$$\delta(q_0, \square) = (ja, \square, L)$$

$$\delta(q_0, a) = (nein, a, N) \quad \text{für } a \neq 0, \square$$

## Definition (LOOP-Programm)

Syntax von **LOOP-Programmen**.

Es ist  $X \in \{x_0, x_1, \dots\}$  und  $C \in \mathbb{N}$ .

```
P → X := X + C
   | X := X - C
   | P; P
   | LOOP X DO P END
   | IF X = 0 DO P ELSE Q END
```

- Ausgabe steht in  $x_0$ , Eingaben in  $x_1, \dots, x_n$ , Rest ist 0.
- **LOOP  $x_i$  DO P END** führt  $P$  genau  $n$  mal aus, wobei  $n$  der Anfangswert von  $x_i$  ist. **Zuweisungen an  $x_i$  in  $P$  ändern die Anzahl der Durchläufe nicht.**

## Definition (Basisfunktionen)

**Primitiv Rekursiv** sind:

- Die konstante Funktion **0**
- Die **Nachfolgerfunktion**  $s(n) = n + 1$
- Die **Projektionsfunktion**  $\pi_i^k : \mathbb{N}^k \mapsto \mathbb{N}, i \in [k]$

$$\pi_i^k(x_1, \dots, x_k) = x_i$$

## Definition (Komposition)

Sind  $g$  und  $h_i$  PR und  $\bar{x} = (x_1, \dots, x_n)$ , dann ist auch  $f$  PR:

$$f(\bar{x}) = g(h_1(\bar{x}), \dots, h_k(\bar{x}))$$

## Basisfunktionen und Komposition

Schon **PR** sind:

- Konstante: 0
- Nachfolger:  $s(n) = n + 1$
- Projektion:  $\pi_j^k : \mathbb{N}^k \mapsto \mathbb{N}$
- Komposition:  $f(\bar{x}) = g(h_1(\bar{x}), \dots, h_k(\bar{x}))$

## Definition (Primitive Rekursion)

Das Schema der **primitiven Rekursion** erzeugt aus  $g$  und  $h$  die Funktion  $f$ :

$$f(0, \bar{x}) = g(\bar{x})$$

$$f(m + 1, \bar{x}) = h(f(m, \bar{x}), m, \bar{x})$$



U.a. diese Programme sind laut Vorlesung oder Übung PR:

- $add(x, y) = x + y$
- $mult(x, y) = x \cdot y$
- $pred(x + 1) = \max \{0, x\}$
- $x \dot{-} y = \max \{0, x - y\}$
- $div(x, y) = x \div y$  (Ganzzahldivision)
- $mod(x, y) = x \bmod y$
  
- $tower(n) = 2^{2^{2^{\dots}}}$  mit  $tower(4) = 2^{16}$
- $sqr(x) = x^2$
- $twopow(n) = 2^n$
- $ifthen(n, a, b) = \begin{cases} a & n = 0 \\ b & n \neq 0 \end{cases}$

## Definition (Erweitertes PR-Schema)

Das **erweiterte Schema der primitiven Rekursion** erlaubt

$$\begin{aligned}f(0, \bar{x}) &= t_0 \\ f(m+1, \bar{x}) &= t\end{aligned}$$

wobei

- $t_0$  enthält nur PR-Funktionen und die  $x_j$
- $t$  enthält nur  $f(m, \bar{x})$ , PR Funktionen,  $m$  und die  $x_j$ .

## Satz

*Das erweiterte Schema der primitiven Rekursion führt nicht aus **PR** heraus.*

## Definition (WHILE-Programm)

Syntax von **WHILE-Programmen**.

Es ist  $X \in \{x_0, x_1, \dots\}$  und  $C \in \mathbb{N}$ .

```
P → X := X + C
   | X := X - C
   | P; P
   | WHILE X ≠ 0 DO P END
   | LOOP X DO P END
   | IF X = 0 DO P ELSE Q END
```

- Ausgabe steht in  $x_0$ , Eingaben in  $x_1, \dots, x_n$ , Rest ist 0.
- Semantik wie erwartet.

## Definition (GOTO-Programm)

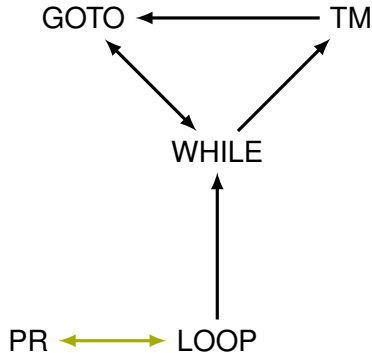
Syntax von **GOTO-Programmen**.

Es ist  $X \in \{x_0, x_1, \dots\}$  und  $C \in \mathbb{N}$ .

Alle Anweisungen haben eine Markierung  $M_1 : A_1; M_2 : A_2$ .

```
P → X := X + C
   | X := X - C
   | P; P
   | GOTO Mi
   | IF X = 0 GOTO Mi
   | HALT
```

- Ausgabe steht in  $x_0$ , Eingaben in  $x_1, \dots, x_n$ , Rest ist 0.



LOOP kann in WHILE **übersetzt** werden, WHILE ist also **mindestens so mächtig** wie LOOP (sogar mächtiger).