Typen von Programmiersprachen

- Berechenbarkeitstheorie: Formalisierung des intuitiven Berechenbarkeitsbegriffs

- man kann vier Typen von Programmiersprachen zum Berechnen von Zahlenfunktionen unterscheiden:

  Programmiersprache LOOP: enthält als wesentliches Element Schleifen mit fester Durchlaufzahl

  Programmiersprache WHILE: enthält als wesentliches Element Schleifen mit bedingungsabhängiger Durchlaufzahl

  Programmiersprache GOTO: enthält als wesentliches Element bedingte Sprungbefehle

  Programmiersprache RECUR: enthält als wesentliches Element rekursive Funktionsdefinitionen

- offensichtlich sind diese Sprachen unterschiedlich mächtig

- Mächtigkeit der Sprachen drückt sich durch die Funktionen aus, die man mit ihnen berechnen kann

- Welche Funktionen sind das?

Elementare Funktionen

- Nullfunktion

  $Z: \mathbf{N}_0 \rightarrow \mathbf{N}_0$

  mit $\forall n \in \mathbf{N}_0 \; (Z(n) := 0)$

- Nachfolgerfunktion

  $S: \mathbf{N}_0 \rightarrow \mathbf{N}_0$

  mit $\forall n \in \mathbf{N}_0 \; (S(n) := n'=n+1)$

- Projektionsfunktion

  $\forall m,k \in \mathbf{N}, \; k \leq m \; (P_k^m: \mathbf{N}_0^m \rightarrow \mathbf{N}_0)$

  mit $\forall x_1,...,x_m \in \mathbf{N}_0 \; (P_k^m(x_1,...,x_m) := x_k)$

Primitiv rekursive Funktionen

- primitiv rekursive Funktionen sind eine induktiv definierte Teilmenge aller Funktionen $\mathbf{N}_0^{\ n} \rightarrow \mathbf{N}_0$ mit $n \in \mathbf{N}$

elementare Funktionen sind primitiv rekursiv
Nullfunktion, Nachfolgerfunktion, Projektion

Komposition
sind für $m, n \in \mathbf{N}$ die Funktionen
$g: \mathbf{N}_0^{\ m} \rightarrow \mathbf{N}_0$
$f_i: \mathbf{N}_0^{\ n} \rightarrow \mathbf{N}_0 \qquad i=1, ..., m$
primitiv rekursiv, dann ist auch die Funktion
$r: \mathbf{N}_0^{\ n} \rightarrow \mathbf{N}_0$
$r(x_1,..., x_n) := g(f_1(x_1,..., x_n), ..., f_m(x_1,..., x_n))$
primitiv rekursiv

primitive Rekursion
sind für $n \in \mathbf{N}_0$ die Funktionen
$g: \mathbf{N}_0^{\ n} \rightarrow \mathbf{N}_0$
$h: \mathbf{N}_0^{\ n+2} \rightarrow \mathbf{N}_0$
primitiv rekursiv, dann ist auch die Funktion
$r: \mathbf{N}_0^{\ n+1} \rightarrow \mathbf{N}_0$
$r(x_1,..., x_n, 0) := g(x_1,..., x_n)$
$r(x_1,..., x_n, y+1) := h(x_1,..., x_n, y, r(x_1,..., x_n, y))$
primitiv rekursiv

Primitiv rekursive Funktionen

- Fall n=0:

  $g \in \mathbf{N}_0$ eine Konstante, $h: \mathbf{N}_0^2 \rightarrow \mathbf{N}_0$ primitiv rekursiv, dann ist $r: \mathbf{N}_0 \rightarrow \mathbf{N}_0$ primitiv rekursiv

  $r(0) = g$

  $r(y+1) = h(y, r(y))$

- Beispiel

  $g=0$, $h(x,y) = x$

  $r(0) = g = 0$

  $r(y+1) = h(y, r(y)) = y$

  d.h. $r(y) = y-1$ (Vorgängerfunktion für $y>0$)

- folgende weitere Funktionen sind primitiv rekursiv

  Addition

  Subtraktion

  Multiplikation

  Exponentialfunktion

  Signumfunktion

  Fakultät

- Funktionen sind genau dann primitiv rekursiv, wenn sie durch ein LOOP Programm berechnet werden können

- Sind alle Funktionen primitiv rekursiv?

Ackermann Funktion

- Ackermann Funktion A: $\mathbf{N}_0^{\,2} \rightarrow \mathbf{N}_0$

  $A(0, 0) = 1$

  $A(0, 1) = 2$

  $A(0, y+2) = y+4$

  $A(x+1, 0) = A(x, 1)$

  $A(x+1, y+1) = A(x, A(x+1, y))$

- verschiedene Autoren definieren leicht abweichende Versionen der Ackermann Funktion

- Ackermann Funktion ist – anders als die primitiv rekursiven Funktionen – rein rekursiv definiert

- Ackermann Funktion ist nicht primitiv rekursiv, sondern gehört in die Klasse der partiell rekursiven Funktionen

Partiell rekursive Funktionen

- partiell rekursive Funktionen sind eine induktiv definierte Teilmenge aller partiellen Funktionen $\mathbf{N}_0^{\ n} \rightarrow \mathbf{N}_0$ mit $n \in \mathbf{N}$

elementare Funktionen sind partiell rekursiv

Nullfunktion, Nachfolgerfunktion, Projektion

Komposition

Komposition partiell rekursiver Funktionen ist partiell rekursiv

primitive Rekursion

primitive Rekursion partiell rekursiver Funktionen liefert wieder partiell rekursive Funktionen

Minimierung

ist für $n \in \mathbf{N}$ die Funktion f: $\mathbf{N}_0^{\ n+1} \rightarrow \mathbf{N}_0$ partiell rekursiv, dann auch die Funktion

$\mu(f) : \mathbf{N}_0^{\ n} \rightarrow \mathbf{N}_0$

Dabei gilt:

$[\mu(f)](x_1,..., x_n) = x$

wobei x die kleinste Lösung von $f(x_1,..., x_n, x) = 0$ ist; sonst ist $[\mu(f)](x_1,..., x_n)$ undefiniert

Church These

- Funktionen sind genau dann partiell rekursiv, wenn sie durch ein WHILE oder durch ein GOTO oder durch ein RECUR Programm berechnet werden können.

- in keiner Programmiersprache kann man mehr Funktionen berechnen als die partiell rekursiven

- These von Church

  Eine Funktion kann genau dann "nach vorgegebenen Regeln" berechnet werden, wenn sie partiell rekursiv ist, d.h. es gibt keine weiteren berechenbaren Funktionen.

  These von Church verbindet den intuitiven Begriff der Berechenbarkeit mit einer formalen Definition.

Programmiersprachen

- WHILE, GOTO und RECUR Programmiersprachen sind gleich mächtig

- Grundlage für strukturierte Programmierung

  Substituierung von (bedingten) Sprüngen durch strukturierte Konstrukte für Sequenz, Iteration und Selektion

- Grundlage für die Substituierung von Rekursion durch Iteration

  Speicherplatz wird eingespart: Rekursion braucht für jeden Schritt zusätzlichen Speicherplatz, während Iteration Speicherplatz wieder verwenden kann

- neben der Mächtigkeit gibt es aber noch weitere Kriterien für die Wahl einer Programmiersprache, z.B.

  Ausdrucksfähigkeit: wie direkt kann man die Konzepte des Anwendungsbereiches in Konstrukten der Programmiersprache ausdrücken, wie lesbar ist das Programm?

  Strukturierungsmöglichkeiten, Wiederverwendung

  effiziente Implementierung

Turingmaschinen

- Beispiel zeigte, dass Turingmaschinen zur Berechnung verwendet werden können

- partielle Funktion f: $(A^*)^n \rightarrow (A^*)^m$ heisst Turing-berechenbar, wenn es eine Turingmaschine mit dem Eingabealphabet $A \cup \{\%,\#\}$ gibt, die mit dem Bandinhalt $w_1\% \; w_2\% \; ...\% w_n$ startet, genau dann hält, wenn $(w_1, w_2, ..., w_n)$ im Definitionsbereich von f liegt und den Bandinhalt $u_1\%u_2\%...\%u_m$ schreibt.

  Es gilt dann $f(w_1,w_2, ...,w_n) = (u_1,u_2,...,u_m)$

- Funktionen sind genau dann Turing-berechenbar, wenn sie partiell rekursiv sind

Berechenbarkeit und formale Sprachen

- Theorie der Berechenbarkeit kann von natürlichen Zahlen auf formale Sprachen ausgedehnt werden

- Gödelisierung: Wörter einer formalen Sprache werden als natürliche Zahlen kodiert

  A endliches Alphabet

  Gödelisierung über A ist eine linkseindeutige (injektive) Turing-berechenbare Funktion $G: A^* \rightarrow \mathbf{N_0}$

- Standard-Gödelisierung:

  A endliches Alphabet

  $\alpha: A \rightarrow \{1, 2, \ldots, \#A\}$ eine linkseindeutige Funktion

  p_n sei die n-te Primzahl ($p_1 = 2$, $p_2=3$, $p_3=5$, ...)

  Gödelisierung : $A^* \rightarrow \mathbf{N_0}$ mit

  $G(\varepsilon) := 0$

  $G(a_1 a_2 .. a_k) := p_1^{\alpha(a_1)} p_2^{\alpha(a_2)} \ldots p_k^{\alpha(a_k)}$

  Beispiel: $G(\text{ende}) = 2^5 * 3^{14} * 5^4 * 7^5$