Theoretische Informatik II

Einheit 6.3

Rekursive Funktionen

1. Primitiv- und μ-rekursive Funktionen
2. Analyse und Programmierung
3. Äquivalenz zu Registermaschinen

Mathematischer Funktionenkalkül auf $\mathbb{N}$

- **Funktionen entstehen durch Anwendung von Operationen auf Grundfunktionen**
  - Funktionsdefinition benötigt keine Funktionsargumente
  - Das informatiktypische "Baukastensystem" entspricht dieser Idee
- **Bausteine gelten als intuitiv berechenbar**
  - Grundfunktionen: Konstante, Projektion, Nachfolgerzahl
  - Operationen: Komposition, einfache Rekursion, Suchschleife
- **Berechnung durch schrittweise Auswertung**
  - Direkte Auswertung von Argumenten bei Grundfunktionen
  - Einsetzen des Definitionsschemas bei Operationen

Bausteine μ-rekursiver Funktionen

1. **Nachfolgerfunktion** $\boldsymbol{s}:\mathbb{N}\to\mathbb{N}$ mit $s(x) = x+1$ für alle $x\in\mathbb{N}$

2. **Projektionsfunktionen** $\boldsymbol{pr_k^n}:\mathbb{N}^n\to\mathbb{N}$ mit $pr_k^n(x_1,\dots,x_n) = x_k$ $(1\leq k\leq n)$

3. **Konstantenfunktion** $\boldsymbol{c_k^n}:\mathbb{N}^n\to\mathbb{N}$ mit $c_k^n(x_1,\dots,x_n) = k$ $(0\leq n)$

4. **Komposition** $\boldsymbol{f\circ(g_1\dots g_n)}:\mathbb{N}^k\to\mathbb{N}$ Definition K

   $h = f\circ(g_1\dots g_n)$, wenn $h(\vec{x}) = f(\,g_1(\vec{x}),\dots,g_n(\vec{x})\,)$

   h entsteht aus $f:\mathbb{N}^n\to\mathbb{N}$ und $g_1\dots g_n:\mathbb{N}^k\to\mathbb{N}$ durch simultane Einsetzung

5. **Primitive Rekursion** $\boldsymbol{Pr[f,g]}:\mathbb{N}^k\to\mathbb{N}$ Definition L.1

   $h = Pr[f,g]$, wenn $h(\vec{x},0) = f(\vec{x})$, $h(\vec{x},y+1) = g(\vec{x},y,\,h(\vec{x},y))$

   h entsteht aus $f:\mathbb{N}^{k-1}\to\mathbb{N}$ und $g:\mathbb{N}^{k+1}\to\mathbb{N}$ durch primitive Rekursion

6. **μ-Operator** $\boldsymbol{\mu f}:\mathbb{N}^k\to\mathbb{N}$ Definition L.2

   $$h = \mu f, \text{ wenn } h(\vec{x}) = \begin{cases} min\{y \mid f(\vec{x},y)=0\} & \text{falls dies existiert und alle } f(\vec{x},i), i<y \text{ definiert} \\ \perp & \text{sonst} \end{cases}$$

   h entsteht aus $f:\mathbb{N}^{k+1}\to\mathbb{N}$ durch Minimierung

Abweichende Notation im Skript: $\boldsymbol{N}$ **statt** $\boldsymbol{s}$, $\boldsymbol{\pi_k^n}$ **statt** $\boldsymbol{pr_k^n}$, $\mathbb{N}_0$ **statt** $\mathbb{N}$

Operationen entsprechen Programmstrukturen

- **Komposition $\hat{=}$ Folge von Anweisungen**

  $\mathtt{y}_1$ `:=` $\mathtt{g}_1$`(`$\mathtt{x}_1$`,..,`$\mathtt{x}_m$`);`

  $\vdots$

  $\mathtt{y}_n$ `:=` $\mathtt{g}_n$`(`$\mathtt{x}_1$`,..,`$\mathtt{x}_m$`);`

  `h  := f(`$\mathtt{y}_1$`,..,`$\mathtt{y}_n$`)` $\qquad (\mathtt{h} = h(x_1, .., x_m))$

- **Primitive Rekursion $\hat{=}$ Zählschleife**

  `h := f(`$\mathtt{x}_1$`,..,`$\mathtt{x}_n$`);`

  `for i:=1 to y do h := g(`$\mathtt{x}_1$`,..,`$\mathtt{x}_n$`,i-1,h) fi` $\qquad (\mathtt{h} = h(x_1, .., x_n, y))$

  – Primitive Rekursion arbeitet in umgekehrter Reihenfolge

- **Minimierung $\hat{=}$ While-schleife (unbegrenzte Suche)**

  `y := 0;`

  `while f(`$\mathtt{x}_1$`..`$\mathtt{x}_n$`,y)`$\neq$`0 do y:=y+1 od;`

  `h := y` $\qquad (\mathtt{h} = h(x_1, .., x_n))$

  – Ergebnis ist Anzahl der Schleifendurchläufe bis zum Erfolg

- $f{:}\mathbb{N}^k{\to}\mathbb{N}$ **primitiv-rekursiv**
  - f ist Nachfolger-, Projektions- oder Konstantenfunktion
  - f entsteht aus primitiv-rekursiven Funktionen durch Komposition oder primitive Rekursion

  $\mathcal{T}_{prim}$: Menge der primitiv-rekursiven Funktionen

- $f{:}\mathbb{N}^k{\to}\mathbb{N}$ **μ-rekursiv**
  - f ist Nachfolger-, Projektions- oder Konstantenfunktion
  - f entsteht aus μ-rekursiven Funktionen durch Komposition, primitive Rekursion oder Minimierung

  $\mathcal{T}_{\mu}$: Menge der μ-rekursiven Funktionen

  $\mathcal{R}_{\mu}$: Menge der totalen μ-rekursiven Funktionen

Beobachtungen und offene Fragen

- **μ-rekursive Funktionen können partiell sein**
- **Offensichtlich gilt: $\mathcal{R}_\mu \subseteq \mathcal{T}_\mu$**
- **Offensichtlich gilt: $\mathcal{T}_{prim} \subseteq \mathcal{R}_\mu$**
  - Alle Grundfunktionen sind total und μ-rekursiv
  - Komposition und Primitive Rekursion erhalten Totalität
- **Sind die Inklusionen echt?**
  - oder könnten zwei Klassen zusammenfallen?
- **Ist eine der Klassen vergleichbar zu $\mathcal{RM}$ bzw. $\mathcal{T}$?**

Diese Fragen müssen noch untersucht werden

- $\boldsymbol{f_1 = Pr[pr_1^1, s \circ pr_3^3]}$ **Was macht f_1?**

- **Stelligeitsanalyse:**

  $pr_1^1{:}\mathbb{N}\to\mathbb{N}$, $pr_3^3{:}\mathbb{N}^3\to\mathbb{N}$, $s \circ pr_3^3{:}\mathbb{N}^3\to\mathbb{N}$ $\quad\mapsto\quad f_1{:}\,\mathbb{N}^2\to\mathbb{N}$

- **Abarbeitungsbeispiel:** $f_1(2,2) = 4$

- **Rekursives Verhalten:**

  $f_1(x,0) = pr_1^1(x) = x$

  $f_1(x,y{+}1) = (s \circ pr_3^3)(x,y,f(x,y)) = s(f(x,y)) = f(x,y){+}1$

Das ist die Rekursionsgleichung der Addition

$$\left.\begin{array}{l} x{+}0 = x \\ x{+}(y{+}1) = (x{+}y){+}1 \end{array}\right\} \mapsto\ \boldsymbol{f_1 = add}{:}\,\mathbb{N}^2\to\mathbb{N} \ \text{ mit } add(n,m) = n{+}m$$

Analyse μ-rekursiver Funktionen

- $\boldsymbol{f_2 = \mu c_1^2}$ $$f_2(x) = \begin{cases} min\{y \mid c_1^2(x,y) = 0\} & \text{falls } y \text{ existiert und alle} \\ & c_1^2(x,i), i{<}y \text{ definiert} \\ \perp & \text{sonst} \end{cases}$$ $$= \begin{cases} min\{y \mid 1 = 0\} & \text{falls dies existiert} \\ \perp & \text{sonst} \end{cases}$$ $$= \perp$$
- $\boldsymbol{f_3 = \mu f_1}$ $$f_3(x) = \begin{cases} 0 & \text{falls } x = 0 \\ \perp & \text{sonst} \end{cases}$$
- $\boldsymbol{f_4 = \mu h}$ mit $h(x,y) = \begin{cases} 0 & \text{falls } x = y \\ \perp & \text{sonst} \end{cases}$ $$f_4(x) = \begin{cases} 0 & \text{falls x=0} \\ \perp & \text{sonst} \end{cases}$$

$h(x,y) = 0$ für $x = y$ aber ist h für $x > 0$ und $y < x$ nicht definiert

“Programmierung” μ-rekursiver Funktionen

- **Vorgängerfunktion $\boldsymbol{p}$** : $\mathbb{N}\to\mathbb{N}$ $\qquad p(n) = n\dot{-}1$

- **Rekursives Verhalten:**
  - $p(0) = 0\dot{-}1 = 0$
  - $p(y{+}1) = (y{+}1)\dot{-}1 = y$

- **Beschreibung durch Primitive Rekursion:**
  - Benötigt: $f{:}\mathbb{N}^0\to\mathbb{N}$ mit $f() = 0$ $\qquad \mapsto f = c_0^0$

    und $g{:}\mathbb{N}^2\to\mathbb{N}$ mit $g(y, p(y)) = p(y{+}1) = y$ $\qquad \mapsto g = pr_1^2$

$\mapsto \quad \boldsymbol{p = Pr[c_0^0,\, pr_1^2]}$

Exkurs: Wie genau muss man sein?

Wiederholung

Ein Beweis ist ein Argument, das den Leser überzeugt

- Nicht notwendig formal oder mit allen Details
- Präzise genug. um Details rekonstruieren zu können
- Knapp genug, um übersichtlich und merkbar zu sein
- Gedankensprünge sind erlaubt, wenn Sie die Materie gut genug verstehen, daß Sie nichts mehr falsch machen können
  ... es reicht nicht, daß Sie es einmal richtig gemacht haben
- Tip: ausführliche Lösungen entwickeln, bis Sie genug Erfahrung haben.
  Für Präsentation zentrale Gedanken aus Lösung extrahieren
- Test: verstehen Ihre Kommilitonen Ihre Lösung und warum sie funktioniert?

Beispiele primitiv-rekursiver Funktionen

- **Subtraktion** $\boldsymbol{sub}: \mathbb{N}^2 \to \mathbb{N}$ $\qquad sub(n,m) = n \dot{-} m$
  - $sub(x,0) = x = pr_1^1(x)$
  - $sub(x,y{+}1) = x \dot{-} (y{+}1) = (x \dot{-} y) \dot{-} 1 = p(x \dot{-} y) = p \circ pr_3^3)(x,y,sub(x,y))$

  $\mapsto \quad sub = Pr[pr_1^1, p \circ pr_3^3]$

- **Multiplikation** $\boldsymbol{mul}: \mathbb{N}^2 \to \mathbb{N}$ $\qquad mul(n,m) = n{*}m$
  - $mul(x,0) = 0 = c_0^1(x)$
  - $mul(x,y{+}1) = mul(x,y){+}x = (add \circ (pr_1^3, pr_3^3))(x,y,mul(x,y))$

  $\mapsto \quad mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$

- **Exponentiierung** $\boldsymbol{exp}: \mathbb{N}^2 \to \mathbb{N}$ $\qquad exp(n,m) = n^m$

  $exp = Pr[c_1^1, (mul \circ (pr_1^3, pr_3^3))]$

- **Fakultät** $\boldsymbol{fak}: \mathbb{N} \to \mathbb{N}$ $\qquad fak(n) = n! = 1{*}2{*}\ldots{*}n$

  $fak = Pr[c_1^0, (mul \circ (s \circ pr_1^2, pr_2^2))]$

- **Signum-Funktion** $\boldsymbol{sign}: \mathbb{N} \to \mathbb{N}$ $\qquad sign(n) = \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{sonst} \end{cases}$

  $sign = Pr[c_0^0, c_1^2]$

Programmiertechniken für rekursive Funktionen

- **Definition durch Fallunterscheidung**

$$h(\vec{x}) = \begin{cases} f(\vec{x}) & \text{falls } test(\vec{x}) = 0 \\ g(\vec{x}) & \text{sonst} \end{cases}$$ (f, g und $test$:$\mathbb{N}^k \to \mathbb{N}$ primitiv-rekursiv)

Wende Signum-Funktion auf Testergebnis an und multipliziere auf

– $h(\vec{x}) = (1 \dot{-} sign(test(\vec{x}))) * f(\vec{x}) + sign(test(\vec{x})) * g(\vec{x})$

$\mapsto$ $h \;=\; add \circ (mul \circ (sub \circ (c_1^1, sign \circ test), f),\; mul \circ (sign \circ test, g))$

- **Generelle Summe** $\boldsymbol{\Sigma_{i=0}^{r} f(\vec{x}, i)}$

Generelles Produkt $\boldsymbol{\Pi_{i=0}^{r} f(\vec{x}, i)}$ (f:$\mathbb{N}^{k+1} \to \mathbb{N}$ primitiv-rekursiv)

– $\sum_{i=0}^{0} f(\vec{x}, i) = f(\vec{x}, 0)$

– $\sum_{i=0}^{y+1} f(\vec{x}, i) = (\sum_{i=0}^{y} f(\vec{x}, i)) + f(\vec{x}, y+1)$

$\mapsto$ $\Sigma f \;=\; Pr[f \circ (pr_1^1, c_0^1),\; add \circ (pr_3^3, f \circ (pr_1^3, s \circ pr_2^3))]$ (für k=1)

$\mapsto$ $\Pi f \;=\; Pr[f \circ (pr_1^1, c_0^1),\; mul \circ (pr_3^3, f \circ (pr_1^3, s \circ pr_2^3))]$

Lösungen für k>1 analog

- **Beschränkte Minimierung**

$$h(\vec{x},t) = \begin{cases} min\{y{\leq}t \mid f(\vec{x},y)=0\} & \text{falls dies existiert} \\ t{+}1 & \text{sonst} \end{cases} \qquad (f{:}\mathbb{N}^{k+1}{\to}\mathbb{N} \in \mathcal{T}_{prim})$$

Rekursives Verhalten:

– $h(\vec{x},0) \quad = \begin{cases} 0 & \text{falls}\, f(\vec{x},0)=0 \\ 1 & \text{sonst} \end{cases} = sign(f(\vec{x},0))$

– $h(\vec{x},t{+}1) = \begin{cases} h(\vec{x},t) & \text{falls } h(\vec{x},t){\leq}t \\ t{+}1 & \text{falls } h(\vec{x},t)=t{+}1 \text{ und } f(\vec{x},t{+}1)=0 \\ t{+}2 & \text{sonst} \end{cases}$

Programmierbar mit Fallunterscheidung und primitiver Rekursion (aufwendiger Ausdruck)

Weitere primitiv-rekursive Funktionen

- **Absolute Differenz** $absdiff: \mathbb{N}^2 \rightarrow \mathbb{N}$ $\qquad absdiff(n,m) = |n-m|$
- **Maximum** $\boldsymbol{max}: \mathbb{N}^2 \rightarrow \mathbb{N}$ $\qquad max(n,m) = \begin{cases} n & \text{falls } n \geq m \\ m & \text{sonst} \end{cases}$
- **Minimum** $\boldsymbol{min}: \mathbb{N}^2 \rightarrow \mathbb{N}$ $\qquad min(n,m) = \begin{cases} m & \text{falls } n \geq m \\ n & \text{sonst} \end{cases}$
- **Division** $\boldsymbol{div}: \mathbb{N}^2 \rightarrow \mathbb{N}$ $\qquad div(n,m) = n \div m$
- **Divisionsrest** $\boldsymbol{mod}: \mathbb{N}^2 \rightarrow \mathbb{N}$ $\qquad mod(n,m) = n \bmod m$
- **Quadratwurzel** $\boldsymbol{sqrt}: \mathbb{N} \rightarrow \mathbb{N}$ $\qquad sqrt(n) = \lfloor \sqrt{n} \rfloor$
- **Logarithmus** $\boldsymbol{ld}: \mathbb{N} \rightarrow \mathbb{N}$ $\qquad ld(n) = \lfloor \log_2 n \rfloor$
- **Größter gemeinsamer Teiler** $\boldsymbol{ggT}: \mathbb{N}^2 \rightarrow \mathbb{N}$
- **Kleinstes gemeinsames Vielfaches** $\boldsymbol{kgV}: \mathbb{N}^2 \rightarrow \mathbb{N}$

Berechenbare Numerierung von Zahlenpaaren

	0	1	2	3	4	5	...
0	0	2	5	9	14	20	...
1	1	4	8	13	19		...
2	3	7	12	18			...
3	6	11	17				...
4	10	16					...
5	15						...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	...

$$\langle x,y\rangle := (x+y)(x+y+1)\div 2 + y$$

"Standard-Tupelfunktion"

- $\langle\rangle{:}\mathbb{N}^2\to\mathbb{N}$ **ist primitiv-rekursiv und bijektiv**
- **Die Umkehrfunktionen** $\pi_i^2 := pr_i^2 \circ \langle\rangle^{-1}$ **sind primitiv-rekursiv**
- $\langle\rangle$ **kann iterativ auf** $\mathbb{N}^k\to\mathbb{N}$ **und auf** $\mathbb{N}^*\to\mathbb{N}$ **fortgesetzt werden**
  - $\langle x,y,z\rangle^3 = \langle x,\langle y,z\rangle\rangle, \ldots,\quad \langle x_1...x_k\rangle^* = \langle k, \langle x_1,..,x_k\rangle^k\rangle$
  - Alle Funktionen sind bijektiv und primitiv-rekursiv
  - Alle Umkehrfunktionen π_i^k und π_i^* sind primitiv-rekursiv
- **Jede rekursive Funktion kann einstellig simuliert werden**
  - Für $f{:}\mathbb{N}^2\to\mathbb{N}$ und $g := f\circ(pi_1^2, pi_2^2)$ gilt $g:\mathbb{N}\to\mathbb{N}$ und $f(x,y) = g(\langle x,y\rangle)$

AUSDRUCKSKRAFT REKURSIVER FUNKTIONEN

$$\mathcal{T}_{prim} \subset \mathcal{R}_\mu \subset \mathcal{T}_\mu = \mathcal{RM} = \mathcal{T}$$

- $\mathcal{T}_{prim} \subseteq \mathcal{R}_\mu \subseteq \mathcal{T}_\mu$ gilt offensichtlich
  - Grundfunktionen und Anwendungen von p.r. Operationen sind total
- $\mathcal{R}_\mu \neq \mathcal{R}_\mu$
  - Nicht alle μ-rekursiven Funktionen sind total (Beispiel: $f_3 = \mu\, add$)
- $\mathcal{T}_{prim} \neq \mathcal{R}_\mu$
  - Primitiv-rekursive Funktionen haben endliche Schachtelungstiefe
  - Unbegrenzte Iteration über Schachtelungstiefe ist intuitiv berechenbar
  - Konkretes Beispiel: Ackermann-Funktion
- $\mathcal{T}_\mu = \mathcal{RM}$

  $\subseteq$: Gebe RM-Unterprogramme für Grundfunktionen und Operationen

  $\supseteq$: Beschreibe RM-Konfigurationsübergänge und Terminierung μ-rekursiv

Schachtelungstiefe

Wie kompliziert ist eine Funktion ?

- **Tiefe $\hat{=}$ Anzahl verschachtelter For-Schleifen**
  – Funktionen ohne Minimierung und primitive Rekursion $\mapsto$ Tiefe 0
  – Komposition mit Funktionen der Tiefe n $\mapsto$ Tiefe n
  – Primitive Rekursion mit Funktionen der Tiefe n $\mapsto$ Tiefe $n+1$

- **Beispiele**
  – Tiefe 1: Addition add, Vorgänger p, Signum $sign$
  – Tiefe 2: Multiplikation mul, Subtraktion sub
  – Tiefe 3: Exponentiation exp, Fakultät fak

- **Primitiv-rekursiven Funktionen haben eine begrenzte Schachtelungstiefe**
  – Minimierung ist nicht begrenzbar

Ackermann-Funktionen (1928)

Funktionen mit Schachtelungstiefe n

- **Definiere Funktionen A_n iterativ:**

$$
\begin{aligned}
A_0(x) &:= \begin{cases} 1 & \text{falls } x = 0 \\ 2 & \text{falls } x = 1 \\ x+2 & \text{sonst} \end{cases} \\
A_{n+1}(0) &:= 1 \\
A_{n+1}(x+1) &:= A_n(A_{n+1}(x))
\end{aligned}
$$

- **Jede der Funktionen A_n ist primitiv-rekursiv**

- **Wachstumsverhalten**

$A_1(x) = 2x \; (x \geq 1)$

$A_2(x) = 2^x$

$A_3(x) = \underbrace{2^{(2^{(2^{\cdots^{2}})})}}_{x-mal}$

$A_4(0) = 1$

$A_4(1) = 2$

$A_4(2) = 2^2 = 4$

$A_4(3) = 2^{2^{2^2}} = 65536$

$A_4(4) = \underbrace{2^{(2^{(2^{\cdots^{2}})})}}_{65536-mal}$

$A_4(5) = \underbrace{2^{(2^{(2^{\cdots^{2}})})}}_{A_4(4)-mal}$

Die grosse Ackermann-Funktion

Beispiel für $\mathcal{T}_{prim} \neq \mathcal{R}_\mu$ Satz O

- **Definiere $A(x) := A_x(x)$ (Große Ackermann-Funktion)**
  - Die Berechnung von $A(x)$ benötigt Schachtelungstiefe x
  - Keine Begrenzung der Schachtelungstiefe möglich

- **A kann nicht primitiv-rekursiv sein**
  - A wächst schneller als jede primitiv-rekursive Funktion
  - Für jede p.r. Funktion f gibt es ein k mit $f(n)<A(n)$ für alle $n>k$
    (sehr aufwendig)

- **A ist total und μ-rekursiv**
  - Beschreibe Abarbeitungsfunktion δ eines Berechnungsstacks für A:
    - $\delta\langle wn0\rangle^*=w1$, $\delta\langle w01\rangle^*=w2$, $\delta\langle w0(x+2)\rangle^*=w(x+4)$,
    - $\delta\langle w(n+1)(x+1)\rangle^*=\langle wn(n+1),x\rangle^*$
    - δ ist primitiv-rekursiv (aufwendig)
  - Für $n\in\mathbb{N}$ berechne $A(n) = \delta^k\langle nn\rangle$ für $k = min\{j|\pi_1^2(\delta^j\langle nn\rangle) = 1\}$
    - k existiert immer

$\mathcal{T}_\mu \subseteq \mathcal{RM}$ Satz N.1

- **Simuliere Grundfunktionen und Operationen**
  - Verwende Standardanfangskonfiguration α und -ausgabefunktion ω
  - RM-Unterprogrammtechnik vermeidet Konflikte durch Umbenennung
  - RM-Programmiersprache vereinfacht Beschreibung

- **Simulation der Grundfunktionen**
  - Nachfolgerfunktion p: $\texttt{r}_1 \texttt{ := } \texttt{r}_1\texttt{+1}$
  - Projektionsfunktion pr_k^n: $\texttt{r}_1 \texttt{ := } \texttt{r}_k$
  - Konstantenfunktion c_k^n: $\texttt{r}_1 \texttt{ := } k$

- **RM-Unterprogramm für Komposition $f \circ (g_1...g_n)$**
  - $F, G_1, .., G_n$ RM-Unterprogramme für $f{:}\mathbb{N}^n{\to}\mathbb{N}$ und $g_1...g_n{:}\mathbb{N}^k{\to}\mathbb{N}$, $m{>}max(k,n)$

$$
\begin{aligned}
&\texttt{r}_{m+1} \texttt{ := } G_1(\texttt{r}_1\texttt{,..,}\texttt{r}_k)\\
&\quad\vdots\\
&\texttt{r}_{m+n} \texttt{ := } G_n(\texttt{r}_1\texttt{,..,}\texttt{r}_k)\\
&\texttt{r}_1 \texttt{ := } \texttt{r}_{m+1}\texttt{; ..; } \texttt{r}_n \texttt{ := } \texttt{r}_{m+n}\\
&\texttt{r}_1 \texttt{ := } F(\texttt{r}_1\texttt{,..,}\texttt{r}_n)
\end{aligned}
$$

$\mathcal{T}_\mu \subseteq \mathcal{RM}$: RM-Unterprogramme für Operationen

- **Unterprogramm für Primitive Rekursion $\boldsymbol{Pr[f,g]}$**
  - F, G RM-Unterprogramme für $f{:}\mathbb{N}^{k-1}{\to}\mathbb{N}$ und $g{:}\mathbb{N}^{k+1}{\to}\mathbb{N}$

```
r_{k+1} := F(r_1,..,r_{k-1})
r_{k+2} := r_k;                                             Grenze der Zählschleife
r_k := 0;                                                   Zähle vorwärts
while r_k<r_{k+2} do r_{k+1} := G(r_1,..,r_{k+1}); r_k := r_k+1 od;
r_1 := r_{k+1}
```

- **RM-Unterprogramm für Minimierung $\boldsymbol{\mu f}$**
  - F RM-Unterprogramm für $f{:}\mathbb{N}^{k+1}{\to}\mathbb{N}$, $\quad m{>}max(k,n)$

```
r_{k+1} := 0
r_{k+2} := F(r_1,..,r_{k+1})
while r_{k+2}>0 do r_{k+1} := r_{k+1}+1; r_{k+2} := F(r_1,..,r_{k+1}) od;
r_1 := r_{k+1}
```

$\mathcal{T}_\mu \supseteq \mathcal{RM}$

Satz N.2

- **Codiere RM-Konfigurationen als Zahlentupel**
  - Sei $\rho = (S, k, \delta, s_0, F)$ und o.B.d.A $F{=}\{s_x\}$
  - Codiere $\kappa = (s_j, (i_1,..,i_k))$ als $\bar\kappa{=}(j, i_1,..,i_k)$
- **Simuliere Überführungsfunktion $\hat\delta$**
  - Für $\bar\kappa{=}(j, i_1,..,i_k)$ mit $\delta(s_j, (\text{sign}(i_1),..,\text{sign}(i_k))) = (s_{j'}, (op_1,..,op_k))$ ist

$$\bar\delta(\bar\kappa){=}(j', (i'_1,..,i'_k)), \quad \text{wobei } i'_j = \begin{cases} 0 & \text{falls } i_j{=}\mathtt{0} \text{ und } op_j{=}\mathtt{-1}, \\ i_j{+}op_j & \text{sonst} \end{cases}$$

$\bar\delta$ und die Iteration $\bar\delta^*$ sind primitiv-rekursiv

- **Beschreibe Rechenzeitfunktion t_ρ**
  - $t_\rho(n) = \min\{j \mid pr_1^{k+1}(\bar\delta^*((0, n, 0, .., 0), j)) = x\}$ $\quad$ t_ρ ist μ-rekursiv
- **Beschreibe Semantik h_ρ von ρ**
  - $h_\rho(n) = pr_2^{k+1}(\bar\delta^*((0, n, 0, .., 0), t_\rho(n))$ $\quad$ h_ρ ist μ-rekursiv

Konsequenzen

Für Argumente zur Berechenbarkeit können wahlweise Turingmaschinen Registermaschinen oder μ-rekursive Funktionen eingesetzt werden

Kleene Normalform Theorem:

Für jede berechenbare Funktion h kann man primitiv-rekursive Funktionen f und g konstruieren, so daß

$$h(x) = g(x, \mu f(x))$$

– Konstruiere RM für h und wähle $f = t_\rho$ und $g = pr_2^{k+1} \circ \bar{\delta}^*$

Berechenbare Funktionen kommen mit einer einzigen Minimierung (While-Schleife) aus

Funktionsdefinition ohne primitive Rekursion

- f:$\mathbb{N}^k \rightarrow \mathbb{N}$ **min-rekursiv**
  - f ist Addition, Nachfolger-, Projektions- oder Konstantenfunktion
  - f entsteht aus min-rekursiven Funktionen durch Komposition oder Minimierung

  $\mathcal{T}_{min}$: Menge der min-rekursiven Funktionen

- $\mathcal{T}_{min} = \mathcal{T}_{\mu}$
  - $\mathcal{T}_{min} \subseteq \mathcal{T}_{\mu}$: Offensichtlich, da $add \in \mathcal{T}_{\mu}$
  - $\mathcal{T}_{min} \supseteq \mathcal{T}_{\mu}$: beschreibe Abarbeitung des Rekursionsstacks
    und suche nach erstem erzeugten Stack der Länge 1
    ähnlich wie bei Berechnung der Ackermann Funktion
    (extrem aufwendig)