Theoretische Informatik II

Einheit 4.3

Rekursive Funktionen

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

Berechenbarkeit ohne Maschinenmodell

Welche Arten von Funktionen sind berechenbar?

- **Einfache Funktionen müssen berechenbar sein**
  - Nachfolgerfunktion: von einer Zahl zur nächsten weiterzählen (s)
  - Projektion: aus einer Gruppe von Werten einen herauswählen (pr_k^n)
  - Konstante: unabhängig von der Eingabe eine feste Zahl ausgeben (c_k^n)
- **Berechenbare Funktionen sind zusammensetzbar**
  - Einfache Operationen erzeugen neue berechenbare Funktionen
  - Komposition: Hintereinanderausführen mehrerer Berechnungen
  - Rekursion: Bei der Ausführung ruft eine Funktion sich selbst wieder auf
  - Suche: Bestimmung der ersten Nullstelle einer Funktion
- **Beschreibung benötigt keine Funktionsargumente**
  - Es reicht, die Grundfunktionen und Operationen zu benennen
  - Programmiersprache ist abstrakt: wende Operationen auf Funktionen an

Mathematischer Kalkül für informatiktypisches 'Baukastensystem'

Beispiele für Baukastenprogrammierung

- **Addition von 2**
  - Ergebnis entsteht durch zweifache Addition von 1 $\quad x+2 = x+1+1$
  - Abstrahiert: doppelte Anwendung der Nachfolgerfunktion $\quad x+2 = \boldsymbol{s}(\boldsymbol{s}(x))$
  - Die Funktion "Addition von 2" ist also die Komposition von s und s
- **Addition zweier Zahlen x und y**
  - Ergebnis entsteht durch y-fache Addition von 1 $\quad x+y = x+\underbrace{1+\ldots+1}_{y-mal}$
  - Iteration muß rekursiv beschrieben werden
    - Im Basisfall (y=0) ist das Ergebnis x $\quad x+0 = x$
    - Im Schrittfall verwenden wir die Addition von x und y, um die Addition von x und y+1 zu bestimmen $\quad x+(y+1) = (x+y)+1$
  - Abstrahiert:
    - Basisfall ergibt sich durch Anwendung der Projektion $\boldsymbol{pr}_1^1$ auf x
    - Schrittfall ergibt sich durch Anwendung von $\boldsymbol{s}$ auf das "alte" Ergebnis
- **Subtraktion zweier Zahlen x und y**
  - Suche, von Null beginnend, die kleinste Zahl z mit $z + y = x$

  Suche ist nicht wirklich erforderlich – iterierte Anwendung der Vorgängerfunktion reicht

Bausteine rekursiver Funktionen – präzisiert

- **Grundfunktionen**
  - **Nachfolgerfunktion** $\boldsymbol{s}:\mathbb{N}\to\mathbb{N}$ mit $s(x) = x+1$
  - **Projektionsfunktionen** $\boldsymbol{pr^n_k}:\mathbb{N}^n\to\mathbb{N}$ $(1\leq k\leq n)$ mit $pr^n_k(x_1,..,x_n) = x_k$
  - **Konstantenfunktion** $\boldsymbol{c^n_k}:\mathbb{N}^n\to\mathbb{N}$ $(0\leq n)$ mit $c^n_k(x_1,..,x_n) = k$
- **Operationen auf Funktionen**
  - **Komposition** $\boldsymbol{f\circ(g_1,..,g_n)}:\mathbb{N}^k\to\mathbb{N}$ $(g_1,..,g_n{:}\mathbb{N}^k\to\mathbb{N},\ f{:}\mathbb{N}^n\to\mathbb{N})$

    Für $h = f\circ(g_1,..,g_n)$ gilt $h(\vec{x}) = f(\,g_1(\vec{x}),..,g_n(\vec{x})\,)$
  - **Primitive Rekursion** $\boldsymbol{Pr[f,g]}:\mathbb{N}^k\to\mathbb{N}$ $(f{:}\mathbb{N}^{k-1}\to\mathbb{N},\ g{:}\mathbb{N}^{k+1}\to\mathbb{N})$

    Für $h = Pr[f,g]$ gilt $h(\vec{x},0) = f(\vec{x})$,

    und $h(\vec{x},y{+}1) = g(\vec{x},y,\ h(\vec{x},y))$
  - μ**-Operator (Minimierung)** $\boldsymbol{\mu f}:\mathbb{N}^k\to\mathbb{N}$ $(f{:}\mathbb{N}^{k+1}\to\mathbb{N})$

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

    Notation auch: $\boldsymbol{h(x) = \mu_z[f(x,z) = 0]}$

Operationen entsprechen Programmstrukturen

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

```
y_1 := g_1(x_1,..,x_k);
   ⋮
y_n := g_n(x_1,..,x_k);
h   := f(y_1,..,y_n)
```

($\mathtt{h} \hat{=} h(x_1,..,x_k)$ für $h = f \circ (g_1,..,g_n)$)

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

```
h := f(x_1,..,x_k);
for i:=1 to y do
   h := g(x_1,..,x_k,i-1,h)
od
```

($\mathtt{h} \hat{=} h(x_1,..,x_k,y)$ für $h = Pr[f,g]$)

- **Minimierung $\hat{=}$ Suche mit While-schleife**

```
y := 0;
while f(x_1,..,x_k,y)≠0 do
   y:=y+1
od;
h := y
```

(unbegrenzte Suche)

($\mathtt{h} \hat{=} h(x_1,..,x_k)$ für $h = \mu f$)

Primitiv- und μ-rekursive Funktionen

- **$\mathcal{PR}$: Menge der primitiv-rekursiven Funktionen**
  - Nachfolger-, Projektions- oder Konstantenfunktion sowie
  - Alle Funktionen, die aus primitiv-rekursiven Funktionen durch Komposition oder primitive Rekursion entstehen

  Primitiv-rekursive Funktionen sind total (terminieren immer)
- **$\mathcal{R}$: Menge der (μ-)rekursiven Funktionen**
  - Nachfolger-, Projektions- oder Konstantenfunktion sowie
  - Alle Funktionen, die aus μ-rekursiven Funktionen durch Komposition, primitive Rekursion oder Minimierung entstehen

  μ-rekursive Funktionen können partiell sein
- **$\mathcal{TR}$: Menge der totalen rekursiven Funktionen**

Es gilt $\mathcal{PR} \subset \mathcal{TR} \subset \mathcal{R}$

- Teilmengenbeziehung gilt offensichtlich (Beispiele für Unterschiede später)

Beispiel einer primitiv-rekursiven Funktion

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

  $pr_1^1{:}\mathbb{N}\to\mathbb{N}$, $pr_3^3{:}\mathbb{N}^3\to\mathbb{N}$, $s\circ pr_3^3{:}\mathbb{N}^3\to\mathbb{N}$ $\mapsto$ $\boldsymbol{f_1}{:}\,\mathbb{N}^2\to\mathbb{N}$
- **Auswertung durch schrittweises Einsetzen**

  Beispiel: $\boldsymbol{f_1(2,2) = 4}$

  – Einsetzen des Definitionsschemas bei Operationen

  – Direkte Auswertung von Argumenten bei Grundfunktionen
- **Analyse des rekursiven Verhaltens:**

  – $f_1(x,0) = pr_1^1(x) = x$

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

  Das ist die Rekursionsgleichung der Addition

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

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 h ist für $x{>}0$ und $y{<}x$ nicht definiert

‘Programmierung’ μ-rekursiver Funktionen

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

  Finde ‘Programm’ im Kalkül der μ-rekursiven Funktionen

- **Analysiere rekursives Verhalten:**
  - $p(0) \quad = 0 \dot{-} 1 = 0$ (Subtraktion liefert niemals weniger als 0!!)
  - $p(y{+}1) = (y{+}1) \dot{-} 1 = y$

- **Beschreibe Verhalten als Primitive Rekursion:**
  - Benötigt: $f{:}\mathbb{N}^0 \to \mathbb{N}$ mit $p(0) \quad = f() = 0 \qquad \mapsto f = c_0^0$

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

Abstraktes “Programm”: $p = Pr[c_0^0, pr_1^2]$

Beispiele primitiv-rekursiver Funktionen

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

  Abstraktes Programm: $\boldsymbol{sub = Pr[pr_1^1, p \circ pr_3^3]}$

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

  $\boldsymbol{mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]}$

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

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

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

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

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

  $\boldsymbol{sign = Pr[c_0^0, c_1^2]}$

Primitiv-rekursive Programmiertechniken

- **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, $test$:$\mathbb{N}^k \to \mathbb{N}$ primitiv-rekursiv)

Wende Signum-Funktion auf Testergebnis an und multipliziere auf

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

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

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

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

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

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

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

$\Pi f = Pr[f \circ (pr_1^1, c_0^1),\; mul \circ (pr_3^3, f \circ (pr_1^3, s \circ pr_2^3))]$ (Lösungen für k>1 analog)

Primitiv-rekursive Programmiertechniken

- **Beschränkte Minimierung**

$$h(\vec{x},t) = \begin{cases} min\{y{\leq}t \mid f(\vec{x},y) = 0\} & \text{falls dies existiert} \\ t{+}1 & \text{sonst} \end{cases}$$

Schreibweise: $\boldsymbol{h(\vec{x},t) = Mn_t[f](\vec{x})}$

Rekursives Verhalten:

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

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

Programmierbar mit Fallunterscheidung und primitiver Rekursion (aufwendiger Ausdruck)

***h* ist primitiv rekursiv, wenn *f* primitiv rekursiv ist**

Weitere primitiv-rekursive Funktionen

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

Beweise in Übungen selbst durchführen !

Berechenbare Numerierung von Zahlenpaaren

⋮	⋮	⋮	⋮	⋮	⋮	⋮	…
5	20						…
4	14	19					…
3	9	13	18				…
2	5	8	12	17			…
1	2	4	7	11	16		…
0	0	1	3	6	10	15	…
y/x	0	1	2	3	4	5	…

$$\langle x, y\rangle \;:=\; (\Sigma_{i=1}^{x+y} i) + y$$
$$=$$
$$(x+y)(x+y+1) \div 2 + y$$

"**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**
  - $\langle x, y, z\rangle^3 = \langle x, \langle y, z\rangle\rangle, \ldots,\quad \langle x_1 \ldots x_k\rangle^* = \langle k, \langle x_1, .., x_k\rangle^k\rangle$
  - 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 $g\langle x, y\rangle = f(x, y)$

Die Ackermann-Funktion (1928)

- **Definiere Funktionen A_n iterativ:**

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

Jede der Funktionen A_n ist primitiv-rekursiv

- **Wachstumsverhalten**

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

$A_2(x) = 2^x$

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

$A_4(0) = 1$

$A_4(1) = 2$

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

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

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

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

- **Definiere $A(x) := A_x(x)$** (Große Ackermann-Funktion)

A ist nicht primitiv-rekursiv aber μ-rekursiv und total

Warum kann A nicht primitiv-rekursiv sein ?

- **Betrachte Schachtelungstiefe einer Funktion**

  Anzahl der ineinander verschachtelten 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 $n+1$

  Primitiv-rekursive Funktionen haben eine begrenzte Schachtelungstiefe

- **Beispiele**

  – Tiefe 1: Addition add, Vorgänger p, Signum $sign$

  – Tiefe 2: Multiplikation mul, Subtraktion sub

  – Tiefe 3: Exponentiation exp, Fakultät fak

- **Die Schachtelungstiefe von A ist unbegrenzbar**

  – Die Berechnung von $A(x)$ benötigt Schachtelungstiefe x

  – **A kann nicht primitiv-rekursiv sein**

  Präzises formales Argument ist sehr aufwendig

Warum ist die Ackermannfunktion berechenbar?

- **Intuitiver Berechnungsmechanismus einfach**
  - Schrittweise Abarbeitung der Rekursion
  - Abarbeitung des Rekursionsstacks ist 'programmierbar'
- ***A* ist total und *μ*-rekursiv**
  - Beschreibe Abarbeitungsfunktion δ eines Berechnungsstacks für A unter Verwendung der Standardtupelfunktion auf Worten über $\mathbb{N}$

$$\begin{aligned}
\delta\langle wx0\rangle^* &= w1\\
\delta\langle w01\rangle^* &= w2\\
\delta\langle w0(y{+}2)\rangle^* &= w(y{+}4)\\
\delta\langle w(x{+}1)(y{+}1)\rangle^* &= \langle wx(x{+}1)y\rangle^*
\end{aligned}$$

  - δ ist primitiv-rekursiv (Beschreibung des Programms aufwendig)
  - Berechne $A(x) = \delta^k\langle xx\rangle^*$, wobei $k = \mu_j\,[\pi_1^2(\delta^j\langle xx\rangle^*) = 1]$
  - Berechnung terminiert, da in jedem δ-Schritt entweder die erste bearbeitete Zahl im Stack oder die Anzahl der Zahlen kleiner wird

Min-rekursive Funktionen

Funktionsdefinition ohne primitive Rekursion

- $\mathcal{R}_{min}$: **Menge der min-rekursiven Funktionen**
  - Addition, Nachfolger, Projektions- oder Konstantenfunktion sowie
  - Alle Funktionen, die aus min-rekursiven Funktionen durch Komposition oder Minimierung entstehen

  Wichtiger Sonderfall für Vergleiche mit anderen Modellen

- $\mathcal{R}_{min} \subseteq \mathcal{R}$: min-rekursive Funktionen sind μ-rekursiv
  - Offensichtlich, da Addition μ-rekursiv ist

- $\mathcal{R} \subseteq \mathcal{R}_{min}$: μ-rekursive Funktionen sind min-rekursiv
  - Beschreibe Abarbeitung des Stacks einer primitiven Rekursion
  - Suche nach erstem erzeugten Stack der Länge 1 (Details aufwendig)

Ausdruckskraft rekursiver Funktionen

- $\mathcal{R} \subseteq \mathcal{T}$: rekursive Funktionen sind Turing-berechenbar
  - Alle Grundfunktionen sind konventionell berechenbar
  - Komposition, Primitive Rekursion und μ-Operator sind berechenbar
  - Konventionell berechenbare Funktionen sind Turing-berechenbar
- $\mathcal{T} \subseteq \mathcal{R}$: Turing-berechenbare Funktionen sind rekursiv
  - Codiere Konfigurationen (Worttupel) als Zahlentupel
  - Simuliere Konfigurationsübergänge $\vdash$ als primitiv-rekursive Funktionen
  - Beschreibe Terminierung von $\vdash^*$ als Suche nach Endkonfiguration
  - Semantik der Turingmaschine ist Iteration von $\vdash$ bis Terminierung
- **$\mathcal{PR}$ und $\mathcal{TR}$ sind echte Unterklassen von $\mathcal{R}$**
  - $\mathcal{PR} \subseteq \mathcal{TR} \subseteq \mathcal{R}$ gilt offensichtlich
  - $\mathcal{TR} \neq \mathcal{R}$: nicht alle rekursiven Funktionen sind total (z.B. $f_3 = \mu\, add$)
  - $\mathcal{PR} \neq \mathcal{TR}$: die Ackermannfunktion ist total, aber nicht primitiv rekursiv

Konsequenzen der Äquivalenzbeweise

- **Für formale Argumente zur Berechenbarkeit können wahlweise Turingmaschinen 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

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

  – Simuliere Turingmaschine für h als μ-rekursive Funktion
  – f ist die Funktion, die Terminierung charakterisiert
  – μf berechnet die Anzahl der Schritte bis zur Terminierung
  – g berechnet die Iteration der Konfigurationsübergänge

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

Rekursive Funktionen im Rückblick

- **Axiomatische Definition von Berechenbarkeit**
  - Abstrakter Funktionenkalkül ohne direkte Benennung der Argumente
  - Bausteine: Grundfunktionen s, pr_k^n, c_k^n und Operatoren $\circ$, Pr, μ
  - Rekursive Funktionen sind Turing-mächtig
- **Minimierung ist der mächtigste Mechanismus**
  - Oft einfacher und eleganter als Anwendung der primitiven Rekursion
  - Terminierung im Allgemeinfall jedoch nicht gesichert
  - Theoretisch braucht jede rekursive Funktion maximal eine Minimierung
- **Primitiv-rekursive Funktionen als Unterklasse**
  - Abstraktes Programm darf keine Minimierung enthalten
  - Fast alle wichtigen Funktionen sind primitiv-rekursiv
  - Fast alle Programmiertechniken erhalten primitive Rekursivität
  - Es gibt totale berechenbare Funktionen, die nicht primitiv-rekursiv sind

Mehr Details in Lewis & Papadimitriou §4.7 und den auf der Webseite genannten Skripten