

Theoretische Informatik II

Einheit 5.2

Rekursive Funktionen



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

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**: Programm ruft sich bei Ausführung selbst auf
 - **Suche**: nach der ersten Nullstelle einer Funktion
- **Beschreibung benötigt keine Funktionsargumente**
 - **Abstrakte Programmiersprache**: wende Operationen auf Funktionen an
 - Es reicht, die **Grundfunktionen** und **Operationen** zu benennen
 - **Mathematischer Kalkül** für informatiktypisches ‘Baukastensystem’

BEISPIELE FÜR BAUKASTENPROGRAMMIERUNG

● Addition von 2

- Ergebnis entsteht durch zweifache Addition von 1 $x+2 = x+1+1$
- Abstrahiert: doppelte Anwendung der Nachfolgerfunktion $x+2 = s(s(x))$
- Die Funktion $+_2$ ist also die **Komposition** von s und s $+_2 \equiv s \circ s$

● Addition zweier Zahlen x und y

- Ergebnis entsteht durch y -fache Addition von 1 $x+y = x + \underbrace{1 + \dots + 1}_{y\text{-mal}}$
- Iteration muß **rekursiv** beschrieben werden
 - Im Basisfall ($y=0$) ist das Ergebnis x $x+0 = x$
 - Im Schrittfall verwenden wir die Addition von x und y , um die Addition von x und $y+1$ zu bestimmen $x+(y+1) = (x+y)+1$
- Abstrahiert:
 - **Basisfall** ergibt sich durch Anwendung der Projektion pr_1^1 auf x
 - **Schrittfall** ergibt sich durch Anwendung von s auf das “alte” Ergebnis

● kgV zweier Zahlen x und y

- **Suche**, von Null beginnend, die kleinste Zahl z mit x teilt z und y teilt z

PRIMITIV-REKURSIVE FUNKTIONEN

Funktionen, die aus Grundfunktionen durch Kompositionen und/oder primitive Rekursionen entstehen

• Grundfunktionen

- **Nachfolgerfunktion** $s : \mathbb{N} \rightarrow \mathbb{N}$ mit $s(x) = x+1$
- **Projektionsfunktionen** $pr_k^n : \mathbb{N}^n \rightarrow \mathbb{N}$ ($1 \leq k \leq n$) mit $pr_k^n(x_1, \dots, x_n) = x_k$
- **Konstantenfunktion** $c_k^n : \mathbb{N}^n \rightarrow \mathbb{N}$ ($0 \leq n$) mit $c_k^n(x_1, \dots, x_n) = k$

• Operationen auf Funktionen

- **Komposition** $f \circ (g_1, \dots, g_n) : \mathbb{N}^k \rightarrow \mathbb{N}$ ($g_1, \dots, g_n : \mathbb{N}^k \rightarrow \mathbb{N}$, $f : \mathbb{N}^n \rightarrow \mathbb{N}$)

Für $h = f \circ (g_1, \dots, g_n)$ gilt $h(x) = f(g_1(x), \dots, g_n(x))$

- **Primitive Rekursion** $Pr[f, g] : \mathbb{N}^k \rightarrow \mathbb{N}$ ($f : \mathbb{N}^{k-1} \rightarrow \mathbb{N}$, $g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$)

Für $h = Pr[f, g]$ gilt $h(x, 0) = f(x)$,

und $h(x, y+1) = g(x, y, h(x, y))$

In Programmierschreibweise: $h(x, y) = \text{if } y=0 \text{ then } f(x) \text{ else } g(x, y-1, h(x, y-1))$

PROGRAMMIERUNG DER REKURSIVEN OPERATIONEN

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

$Y_1 := g_1(x_1, \dots, x_k) ;$

\vdots

$Y_n := g_n(x_1, \dots, x_k) ;$

$h := f(Y_1, \dots, Y_n)$

(Berechne $h(x_1, \dots, x_k)$ für $h = f \circ (g_1, \dots, g_n)$)

(Das Resultat h entspricht $h(x_1, \dots, x_k)$)

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

$h := f(x_1, \dots, x_k) ;$

for $i := 1$ to y do $h := g(x_1, \dots, x_k, i-1, h)$ od

(Berechne $h(x_1, \dots, x_k, y)$ für $h = Pr[f, g]$)

(Resultat $h \hat{=} h(x_1, \dots, x_k, y)$)

ANALYSE EINER PRIMITIVEN REKURSION

- $f_1 = Pr[pr_1^1, s \circ pr_3^3]$

Was macht f_1 ?

- **Stelligkeitsanalyse:**

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

- **Auswertung durch schrittweises Einsetzen**

Beispiel: $f_1(2, 2) = 4$

- 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

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

PROGRAMMIERUNG MIT PRIMITIVER REKURSION

Konstruiere primitiv-rekursives Programm für Vorgängerfunktion p

• $p : \mathbb{N} \rightarrow \mathbb{N}$ ist definiert durch $p(n) = n - 1$

• **Analysiere rekursives Verhalten der Funktion**

– $p(0) = 0 - 1 = 0$ (Subtraktion liefert niemals weniger als 0!!)

– $p(y+1) = (y+1) - 1 = y$

In Programmierschreibweise: $p(y) = \text{if } y=0 \text{ then } 0 \text{ else } y-1$

• **Beschreibe p im Schema der primitiven Rekursion**

– Die Programmiersprache “rekursive Funktionen” verlangt, daß das Beschreibungsschema $Pr[f, g]$ eingehalten wird

– Benötigt: $f: \mathbb{N}^0 \rightarrow \mathbb{N}$ mit $p(0) = f() = 0 \quad \mapsto f = c_0^0$

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

Abstraktes Programm ist

$$p = Pr[c_0^0, pr_1^2]$$

PROGRAMMIERUNG PRIMITIV-REKURSIVER FUNKTIONEN

- **Subtraktion** $sub : \mathbb{N}^2 \rightarrow \mathbb{N}$ $sub(n, m) = n - m$
 – $sub(x, 0) = x = pr_1^1(x)$
 – $sub(x, y+1) = x - (y+1) = (x - y) - 1 = p(x - y) = (p \circ pr_3^3)(x, y, sub(x, y))$
 Abstraktes Programm: $sub = Pr[pr_1^1, p \circ pr_3^3]$
- **Multiplikation** $mul : \mathbb{N}^2 \rightarrow \mathbb{N}$ $mul(n, m) = n * m$
 – $mul(x, 0) = 0 = c_0^1(x)$
 – $mul(x, y+1) = mul(x, y) + x = (add \circ (pr_1^3, pr_3^3))(x, y, mul(x, y))$
 $mul = Pr[c_0^1, (add \circ (pr_1^3, pr_3^3))]$
- **Exponentiierung** $exp : \mathbb{N}^2 \rightarrow \mathbb{N}$ $exp(n, m) = n^m$
 $exp = Pr[c_1^1, (mul \circ (pr_1^3, pr_3^3))]$
- **Fakultät** $fak : \mathbb{N} \rightarrow \mathbb{N}$ $fak(n) = n! = 1 * 2 * \dots * n$
 $fak = Pr[c_1^0, (mul \circ (s \circ pr_1^2, pr_2^2))]$
- **Vorzeichen-Funktion** $sign : \mathbb{N} \rightarrow \mathbb{N}$ $sign(n) = \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{sonst} \end{cases}$
 $sign = Pr[c_0^0, c_1^2]$

PRIMITIV-REKURSIVE PROGRAMMIERSCHEMATA

- **Programmierung durch Fallunterscheidung**

$$\text{Sei } h(x) = \begin{cases} f(x) & \text{falls } t(x) = 0 \\ g(x) & \text{sonst} \end{cases} \quad (f, g, t: \mathbb{N}^k \rightarrow \mathbb{N})$$

h ist primitiv rekursiv, wenn f , g und t primitiv rekursiv sind

Beweis: wende Vorzeichenfunktion auf $t(x)$ an, multipliziere mit f und g

$$h(x) = (1 - \text{sign}(t(x))) * f(x) + \text{sign}(t(x)) * g(x)$$

Programmschema: $h = \text{add} \circ (\text{mul} \circ (\text{sub} \circ (c_1^k, \text{sign} \circ t), f), \text{mul} \circ (\text{sign} \circ t, g))$

- **Generelle Summe / Produkt: $\sum_{i=0}^r f(x, i)$ / $\prod_{i=0}^r f(x, i)$ primitiv rekursiv, wenn dies für f gilt** ($f: \mathbb{N}^{k+1} \rightarrow \mathbb{N}$)

Beweis: es gilt $\sum_{i=0}^0 f(x, i) = f(x, 0)$

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

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

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

PRIMITIV-REKURSIVE PROGRAMMIERSCHEMATA

- Programmierung durch beschränkte Suche (**Minimierung**)

$$\text{Sei } h(x) = \mathbf{Mn}_g[f] := \begin{cases} \min\{y \leq g(x) \mid f(x, y) = 0\} & \text{falls dies existiert} \\ g(x) + 1 & \text{sonst} \end{cases} \quad (f: \mathbb{N}^{k+1} \rightarrow \mathbb{N}, g: \mathbb{N}^k \rightarrow \mathbb{N})$$

h ist primitiv rekursiv, wenn f und g primitiv rekursiv sind

Beweis: definiere h' durch $h'(x, t) = \begin{cases} \min\{y \leq t \mid f(x, y) = 0\} & \text{falls dies existiert} \\ t + 1 & \text{sonst} \end{cases}$

$$\text{Dann ist } h'(x, 0) = \begin{cases} 0 & \text{falls } f(x, 0) = 0 \\ 1 & \text{sonst} \end{cases} = \text{sign}(f(x, 0))$$

$$\text{und } h'(x, t+1) = \begin{cases} h'(x, t) & \text{falls } h'(x, t) \leq t \\ t+1 & \text{falls } h'(x, t) = t+1 \text{ und } f(x, t+1) = 0 \\ t+2 & \text{sonst} \end{cases}$$

Damit ist h' aus primitiv rekursiven Funktionen mit Fallunterscheidung und primitiver Rekursion konstruierbar und somit selbst **primitiv rekursiv**.

Wegen $h(x) = h'(x, g(x))$ folgt hieraus, daß auch h **primitiv rekursiv** ist

- Beschränkte **Maximierung** $\mathbf{Max}_g[f]$ analog

WEITERE BEISPIELE PRIMITIV-REKURSIVER FUNKTIONEN

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

Beweise in Übungen selbst durchführen !

BERECHNUNGEN AUF ZAHLENPAAREN UND -LISTEN

⋮	⋮	⋮	⋮	⋮	⋮	⋮	...
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	...

$$\langle x, y \rangle := (\sum_{i=1}^{x+y} i) + y$$

$$=$$

$$(x+y)(x+y+1) \div 2 + y$$

“**Standard-Tupelfunktion**”

- $\langle \rangle : \mathbb{N}^2 \rightarrow \mathbb{N}$ 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 $\mathbb{N}^k \rightarrow \mathbb{N}$ und auf $\mathbb{N}^* \rightarrow \mathbb{N}$ fortgesetzt werden
 - $\langle x, y, z \rangle^3 = \langle x, \langle y, z \rangle \rangle, \dots, \langle x_1 \dots x_k \rangle^* = \langle k, \langle x_1, \dots, x_k \rangle^k \rangle$
 - Alle Funktionen sind bijektiv und primitiv-rekursiv
 - Alle Umkehrfunktionen π_i^k und π_i^* sind primitiv-rekursiv

Jede Funktion $f: \mathbb{N}^2 \rightarrow \mathbb{N}$ kann einstellig simuliert werden

- Wähle $g := f \circ (\pi_1^2, \pi_2^2)$, dann gilt $g: \mathbb{N} \rightarrow \mathbb{N}$ und $g(\langle x, y \rangle) = f(x, y)$

REICHEN PRIMITIV-REKURSIVE FUNKTIONEN AUS?

- Definiere **“Ackermann-”**Funktionen A_n iterativ

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

Jede einzelne der Funktionen A_n ist primitiv-rekursiv

- Das Wachstumsverhalten der A_n steigt mit n

$$\begin{array}{lll} A_1(x) = 2x \quad (x \geq 1) & A_4(0) = 1 & A_4(3) = 2^{2^{2^2}} = 65536 \\ A_2(x) = 2^x & A_4(1) = 2 & A_4(4) = \underbrace{2^{(2^{(2^{\dots^2}))}}}_{65536\text{-mal}} \\ A_3(x) = \underbrace{2^{(2^{(2^{\dots^2}))}}}_{x\text{-mal}} & A_4(2) = 2^2 = 4 & A_4(5) = \underbrace{2^{(2^{(2^{\dots^2}))}}}_{A_4(4)\text{-mal}} \end{array}$$

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

- A wächst extrem stark, ist aber **total** (auf jeder Eingabe definiert)
- A ist intuitiv berechenbar **kann aber nicht primitiv-rekursiv sein**

WARUM KANN A NICHT PRIMITIV-REKURSIV SEIN ?

- **Betrachte Schachtelungstiefe einer Funktion**

Anzahl der ineinander verschachtelten Rekursionen

- Funktionen ohne primitive Rekursion \mapsto **Tiefe 0**
- Komposition mit Funktionen der Tiefe n \mapsto **Tiefe n**
- Primitive Rekursion mit Funktionen der Tiefe n \mapsto **Tiefe $n+1$**

Jede primitiv-rekursive Funktion hat eine begrenzte Schachtelungstiefe

- **Beispiele**

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

- **Die Schachtelungstiefe von A ist unbegrenzt**

- Die Berechnung von $A(x)$ benötigt Schachtelungstiefe x
- **A kann nicht primitiv-rekursiv sein**

Präzises formales Argument ist sehr aufwendig (mögliches Projektthema)

WIE KANN MAN DIE ACKERMANNFUNKTION BERECHNEN?

- **Intuitiver Berechnungsmechanismus einfach**

- Schrittweise Abarbeitung der Rekursion
- Verarbeitung von Rekursionsstacks fest in Compilern verankert

- **A ist in im Stil rekursiver Funktionen programmierbar**

(mögliches Projektthema)

- Beschreibe Abarbeitungsfunktion δ eines Berechnungsstacks für A unter Verwendung der Standardtupelfunktion auf Listen über \mathbb{N}

$$\delta\langle wx0 \rangle^* = \langle w1 \rangle^* \quad A_n(x) = 1$$

$$\delta\langle w01 \rangle^* = \langle w2 \rangle^* \quad A_0(x) = 2$$

$$\delta\langle w0(y+2) \rangle^* = \langle w(y+4) \rangle^* \quad A_0(x+2) = x+4$$

$$\delta\langle w(x+1)(y+1) \rangle^* = \langle wx(x+1)y \rangle^* \quad A_{n+1}(x+1) = A_n(A_{n+1}(x))$$

- δ ist primitiv-rekursiv

(Beschreibung des Programms aufwendig)

und $A(x) = \delta^k\langle xx \rangle^*$, wobei k die kleinste Zahl ist mit $\pi_1^2(\delta^j\langle xx \rangle^*)=1$, d.h. der Stack $\delta^k\langle xx \rangle^*$ enthält nur eine Zahl – das Ergebnis

- **Berechnung terminiert**, da in jedem δ -Schritt entweder die erste bearbeitete Zahl im Stack oder die Anzahl der Zahlen kleiner wird

ERWEITERUNG PRIMITIV-REKURSIVER FUNKTIONEN

- **Ackermann Funktion ist mit Suche programmierbar**

- Suche das kleinste k , für das die Eigenschaft $\pi_1^2(\delta^j \langle xx \rangle^*) = 1$ gilt
- Minimierung wie $Mn_g[f]$, aber ohne Obergrenze g für Suchschritte

- **Zusätzliche Operation Minimierung $\mu f : \mathbb{N}^k \rightarrow \mathbb{N}$ ($f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$)**

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

Kurzschreibweise auch: $h(x) = \mu_z[f(x, z) = 0]$ “ μ -Operator”

- Funktionen die aus den Grundfunktionen durch Komposition, primitive Rekursion oder Minimierung entstehen, heißen (μ -)rekursiv

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

$y := 0;$ (Berechne $h(x_1, \dots, x_k)$ für $h = \mu f$)
 $\text{while } f(x_1, \dots, x_k, y) \neq 0 \text{ do } y := y + 1 \text{ od};$ (unbegrenzte Suche)
 $h := y$ (Resultat $h \hat{=} h(x_1, \dots, x_k)$)

ANALYSE μ -REKURSIVER FUNKTIONEN

• $f_2 = \mu c_1^2$

$$f_2(x) = \begin{cases} \min\{y \mid c_1^2(x, y) = 0\} & \text{falls } y \text{ existiert und alle} \\ & c_1^2(x, i), i < y \text{ definiert} \\ \perp & \text{sonst} \end{cases}$$

$$= \begin{cases} \min\{y \mid 1 = 0\} & \text{falls dies existiert} \\ \perp & \text{sonst} \end{cases}$$

$$= \perp$$

• $f_3 = \mu f_1$

$$f_3(x) = \begin{cases} 0 & \text{falls } x = 0 \\ \perp & \text{sonst} \end{cases}$$

• $f_4 = \mu h$, wobei $h(x, y) = 0$, falls $x=y$, und $h(x, y) = \perp$, sonst

$$f_4(x) = \begin{cases} 0 & \text{falls } x = 0 \\ \perp & \text{sonst} \end{cases}$$

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

WICHTIGE KLASSEN REKURSIVER FUNKTIONEN

- **\mathcal{R} : Menge der (μ -)rekursiven Funktionen**
 - Grundfunktionen und alle Funktionen, die hieraus durch (mehrfache) Komposition, primitive Rekursion oder Minimierung entstehen
 - μ -rekursive Funktionen können **partiell** sein (nicht überall definiert)
- **\mathcal{TR} : Menge der totalen rekursiven Funktionen**
 - A ist total rekursiv, aber nicht primitiv-rekursiv
- **\mathcal{PR} : Menge der primitiv-rekursiven Funktionen**
 - Grundfunktionen und alle Funktionen, die hieraus durch (evtl. mehrfache) Komposition oder primitive Rekursion entstehen
 - Primitiv-rekursive Funktionen sind **total** (terminieren immer)

WIE MÄCHTIG SIND REKURSIVE FUNKTIONEN?

- **\mathcal{PR} und \mathcal{TR} sind echte Unterklassen von \mathcal{R}**

- Offensichtlich gilt

$$\mathcal{PR} \subseteq \mathcal{TR} \subseteq \mathcal{R}$$

- Nicht alle rekursiven Funktionen sind total (z.B. $f_3 = \mu \text{ add}$)

$$\mathcal{TR} \neq \mathcal{R}$$

- A ist total rekursiv, aber sind primitiv-rekursiv

$$\mathcal{PR} \neq \mathcal{TR}$$

- **Rekursive Funktionen sind Turing-berechenbar: $\mathcal{R} \subseteq \mathcal{T}$**

- Alle Grundfunktionen sind konventionell programmierbar

- Komposition, Primitive Rekursion und μ -Operator sind programmierbar

- Alle Computerprogramme sind auf Turing-Maschinen simulierbar

- **Turing-berechenbare Funktionen sind rekursiv: $\mathcal{T} \subseteq \mathcal{R}$**

- 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

KONSEQUENZEN DER ÄQUIVALENZBEWEISE

- **In Berechenbarkeitsbeweisen können Turingmaschinen und μ -rekursive Funktionen verwendet werden**
- **Kleene Normalform Theorem:**
 - Für jede berechenbare Funktion h kann man primitiv-rekursive Funktionen f und g konstruieren, so daß $h(x) = g(x, \mu f(x))$
 - Betrachte Simulation der TM 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 können mit einer einzigen unbegrenzten Schleife auskommen**

REKURSIVE FUNKTIONEN IM RÜCKBLICK

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

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