Script generated by TTT

Title: Seidl: Theoretische_Informatik (24.06.2013)

Date: Mon Jun 24 10:16:37 CEST 2013

Duration: 90:51 min

Pages: 97

Ist P primitiv rekursiv, dann auch der beschränkte Existenzquantor

$$\exists x \leq n.\ P(x) \quad =:\ Q(n)$$

denn:

$$\begin{aligned} \hat{Q}(0) &= \hat{P}(0) \\ \hat{Q}(n+1) &= \hat{Q}(n) + \hat{P}(n+1) * (1 \dot{-} \hat{Q}(n)) \\ &= \begin{cases} 1 & \text{falls } P(n+1) \\ \hat{Q}(n) & \text{sonst} \end{cases} \end{aligned}$$

Ist P primitiv rekursiv, dann auch der beschränkte Existenzquantor

$$\exists x \leq n.\ P(x) \quad =:\ Q(n)$$

denn:

$$\begin{aligned} \hat{Q}(0) &= \hat{P}(0) \\ \hat{Q}(n+1) &= \hat{Q}(n) + \hat{P}(n+1) * (1 \dot{-} \hat{Q}(n)) \\ &= \begin{cases} 1 & \text{falls } P(n+1) \\ \hat{Q}(n) & \text{sonst} \end{cases} \end{aligned}$$

Ist P primitiv rekursiv, dann auch der beschränkte Existenzquantor

$$\exists x \leq n.\ P(x) \quad =:\ Q(n)$$

denn:

$$\begin{aligned} \hat{Q}(0) &= \hat{P}(0) \\ \hat{Q}(n+1) &= \hat{Q}(n) + \hat{P}(n+1) * (1 \dot{-} \hat{Q}(n)) \\ &= \begin{cases} 1 & \text{falls } P(n+1) \\ \hat{Q}(n) & \text{sonst} \end{cases} \end{aligned}$$

4.6 PR = LOOP

Hauptproblem bei LOOP $\rightarrow$ PR:
Kodierung *aller* Variablen eines LOOP Programms in *einer* Zahl.

Satz 4.39
Die Cantorsche Paarungsfunktion

$$c(x,y) := \binom{x+y+1}{2} + x = (x+y)(x+y+1)/2 + x$$

ist eine Bijektion zwischen $\mathbb{N}^2$ *und* $\mathbb{N}$.

4.6 PR = LOOP

Hauptproblem bei LOOP $\rightarrow$ PR:
Kodierung *aller* Variablen eines LOOP Programms in *einer* Zahl.

Satz 4.39
Die Cantorsche Paarungsfunktion

$$c(x,y) := \binom{x+y+1}{2} + x = (x+y)(x+y+1)/2 + x$$

ist eine Bijektion zwischen $\mathbb{N}^2$ *und* $\mathbb{N}$.

	0	1	2	3	...
0	0	2	5	9	...
1	1	4	8	13	...
2	3	7	12	18	...
3	6	11	17	24	...
⋮	⋮	⋮	⋮	⋮	⋱

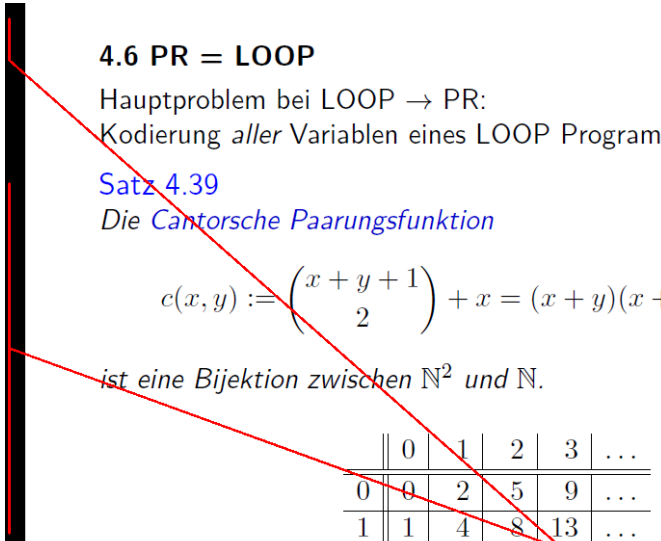
4.6 PR = LOOP

Hauptproblem bei LOOP $\rightarrow$ PR:
Kodierung *aller* Variablen eines LOOP Programms in *einer* Zahl.

Satz 4.39
Die Cantorsche Paarungsfunktion

$$c(x,y) := \binom{x+y+1}{2} + x = (x+y)(x+y+1)/2 + x$$

ist eine Bijektion zwischen $\mathbb{N}^2$ *und* $\mathbb{N}$.



	0	1	2	3	...
0	0	2	5	9	...
1	1	4	8	13	...
2	3	7	12	18	...
3	6	11	17	24	...
⋮	⋮	⋮	⋮	⋮	⋱

4.6 PR = LOOP

Hauptproblem bei LOOP $\to$ PR:
Kodierung *aller* Variablen eines LOOP Programms in *einer* Zahl.

Satz 4.39
Die Cantorsche Paarungsfunktion

$$c(x,y) := \binom{x+y+1}{2} + x = (x+y)(x+y+1)/2 + x$$

ist eine Bijektion zwischen $\mathbb{N}^2$ *und* $\mathbb{N}$.

	0	1	2	3	...
0	0	2	5	9	...
1	1	4	8	13	...
2	3	7	12	18	...
3	6	11	17	24	...
⋮	⋮	⋮	⋮	⋮	⋱

4.6 PR = LOOP

Hauptproblem bei LOOP $\to$ PR:
Kodierung *aller* Variablen eines LOOP Programms in *einer* Zahl.

Satz 4.39
Die Cantorsche Paarungsfunktion

$$c(x,y) := \binom{x+y+1}{2} + x = (x+y)(x+y+1)/2 + x$$

ist eine Bijektion zwischen $\mathbb{N}^2$ *und* $\mathbb{N}$.

	0	1	2	3	...
0	0	2	5	9	...
1	1	4	8	13	...
2	3	7	12	18	...
3	6	11	17	24	...
⋮	⋮	⋮	⋮	⋮	⋱

Die Funktion $x \mapsto \binom{x}{2}$ ist PR:

$$\begin{aligned} \binom{0}{2} &= 0 \\ \binom{n+1}{2} &= \binom{n}{2} + n \end{aligned}$$

Die Funktion $x \mapsto \binom{x}{2}$ ist PR:

$$\begin{aligned} \binom{0}{2} &= 0 \\ \binom{n+1}{2} &= \binom{n}{2} + n \end{aligned}$$

Mit Komposition ist auch c PR:

$$c(x,y) = \binom{x+y+1}{2} + x$$

Mit c kodiert man $k+1$ Tupel:

$$\langle n_0, n_1, \ldots, n_k \rangle \;:=\; c(n_0, c(n_1, \ldots, c(n_k, 0)\ldots))$$

Mit c kodiert man $k+1$ Tupel:

$$\langle n_0, n_1, \ldots, n_k \rangle \;:=\; c(n_0, c(n_1, \ldots, c(n_k, 0)\ldots))$$

Wir brauchen die Umkehrfunktionen p_1 und p_2 von c:

$$p_1(c(x,y)) = x \qquad p_2(c(x,y)) = y$$

Mit c kodiert man $k+1$ Tupel:

$$\langle n_0, n_1, \ldots, n_k \rangle \;:=\; c(n_0, c(n_1, \ldots, c(n_k, 0)\ldots))$$

Wir brauchen die Umkehrfunktionen p_1 und p_2 von c:

$$p_1(c(x,y)) = x \qquad p_2(c(x,y)) = y$$

Damit kann man Projektionsfunktionen auf Tupeln definieren:

$$\begin{aligned} d_0(n) &:= p_1(n) \\ d_1(n) &:= p_1(p_2(n)) \\ &\vdots \\ d_k(n) &:= p_1(\underbrace{p_2 \ldots p_2}_{k}(n)\ldots) \end{aligned}$$

Mit c kodiert man $k+1$ Tupel:

$$\langle n_0, n_1, \ldots, n_k \rangle \;:=\; c(n_0, c(n_1, \ldots, c(n_k, 0)\ldots))$$

Wir brauchen die Umkehrfunktionen p_1 und p_2 von c:

$$p_1(c(x,y)) = x \qquad p_2(c(x,y)) = y$$

Damit kann man Projektionsfunktionen auf Tupeln definieren:

$$\begin{aligned} d_0(n) &:= p_1(n) \\ d_1(n) &:= p_1(p_2(n)) \\ &\vdots \\ d_k(n) &:= p_1(\underbrace{p_2 \ldots p_2}_{k}(n)\ldots) \end{aligned}$$

Sind p_1, p_2 PR, so auch $d_0, \ldots, d_k$.

Satz 4.40

Die Umkehrfunktionen von c sind PR definierbar:

$$\begin{aligned} p_1(n) &= max\{x \leq n \mid \exists y \leq n.\ c(x,y) = n\} \\ p_2(n) &= max\{y \leq n \mid \exists x \leq n.\ c(x,y) = n\} \end{aligned}$$

Satz 4.40

Die Umkehrfunktionen von c sind PR definierbar:

$$\begin{aligned} p_1(n) &= max\{x \leq n \mid \exists y \leq n.\ c(x,y) = n\} \\ p_2(n) &= max\{y \leq n \mid \exists x \leq n.\ c(x,y) = n\} \end{aligned}$$

Beweis:

Alle Funktionen in der Definition der p_i sind PR,

Satz 4.40

Die Umkehrfunktionen von c sind PR definierbar:

$$\begin{aligned} p_1(n) &= max\{x \leq n \mid \exists y \leq n.\ c(x,y) = n\} \\ p_2(n) &= max\{y \leq n \mid \exists x \leq n.\ c(x,y) = n\} \end{aligned}$$

Beweis:

Alle Funktionen in der Definition der p_i sind PR,

auch der Gleichheitstest (Übung!).

Satz 4.40

Die Umkehrfunktionen von c sind PR definierbar:

$$\begin{aligned} p_1(n) &= max\{x \leq n \mid \exists y \leq n.\ c(x,y) = n\} \\ p_2(n) &= max\{y \leq n \mid \exists x \leq n.\ c(x,y) = n\} \end{aligned}$$

Beweis:

Alle Funktionen in der Definition der p_i sind PR,
auch der Gleichheitstest (Übung!).

Die p_i sind die Umkehrfunktionen von c denn:

- Es gibt zu jedem n eindeutige x und y mit $c(x,y) = n$.

Satz 4.40

Die Umkehrfunktionen von c sind PR definierbar:

$$\begin{aligned} p_1(n) &= max\{x \leq n \mid \exists y \leq n.\ c(x,y) = n\} \\ p_2(n) &= max\{y \leq n \mid \exists x \leq n.\ c(x,y) = n\} \end{aligned}$$

Beweis:
Alle Funktionen in der Definition der p_i sind PR,
auch der Gleichheitstest (Übung!).
Die p_i sind die Umkehrfunktionen von c denn:

- Es gibt zu jedem n eindeutige x und y mit $c(x,y) = n$.
  (Bijektivität von c)
- $x \leq c(x,y)$ und $y \leq c(x,y)$.

Damit findet die beschränkte Suche immer die gesuchten x und y. □

Satz 4.41 (PR = LOOP)

Die primitiv rekursiven sind genau die LOOP-berechenbaren Funktionen.

Satz 4.41 (PR = LOOP)

Die primitiv rekursiven sind genau die LOOP-berechenbaren Funktionen.

Beweis:
LOOP $\rightarrow$ PR:

Satz 4.41 (PR = LOOP)

Die primitiv rekursiven sind genau die LOOP-berechenbaren Funktionen.

Beweis:
LOOP $\rightarrow$ PR:
Sei $f : \mathbb{N}^m \rightarrow \mathbb{N}$ LOOP-berechenbar.
Dann gibt es ein LOOP-Programm P (mit Variablen $x_0, \ldots, x_k$),
das f berechnet.

Satz 4.41 (PR = LOOP)

Die primitiv rekursiven sind genau die LOOP-berechenbaren Funktionen.

Beweis:
LOOP $\rightarrow$ PR:
Sei $f : \mathbb{N}^m \rightarrow \mathbb{N}$ LOOP-berechenbar.
Dann gibt es ein LOOP-Programm P (mit Variablen $x_0, \ldots, x_k$), das f berechnet.
Mit Induktion über die Struktur von P
konstruieren wir eine PR Funktion g_P mit

Satz 4.41 (PR = LOOP)

Die primitiv rekursiven sind genau die LOOP-berechenbaren Funktionen.

Beweis:
LOOP $\rightarrow$ PR:
Sei $f : \mathbb{N}^m \rightarrow \mathbb{N}$ LOOP-berechenbar.
Dann gibt es ein LOOP-Programm P (mit Variablen $x_0, \ldots, x_k$), das f berechnet.
Mit Induktion über die Struktur von P
konstruieren wir eine PR Funktion g_P mit

$$g_P(\underbrace{\langle a_0, \ldots, a_k \rangle}) = \underbrace{\langle b_0, \ldots, b_k \rangle}$$

Belegung der Variablen beim Start von P — Belegung der Variablen am Ende von P

Beweis (Forts.):

- P ist x_i := $x_j \pm c$:

Beweis (Forts.):

- P ist x_i := $x_j \pm c$:

$$g_P(x) = \langle d_0(x), \ldots, d_{i-1}(x), d_j(x) \pm c, d_{i+1}(x), \ldots, d_k(x) \rangle$$

Beweis (Forts.):

- P ist x_i := $x_j \pm c$:

$$g_P(x) = \langle d_0(x), \ldots, d_{i-1}(x), d_j(x) \pm c, d_{i+1}(x), \ldots, d_k(x) \rangle$$

- P ist $Q;R$:

Beweis (Forts.):

- P ist x_i := $x_j \pm c$:

$$g_P(x) = \langle d_0(x), \ldots, d_{i-1}(x), d_j(x) \pm c, d_{i+1}(x), \ldots, d_k(x) \rangle$$

- P ist $Q;R$: $g_P(x) = g_R(g_Q(x))$

Beweis (Forts.):

- P ist x_i := $x_j \pm c$:

$$g_P(x) = \langle d_0(x), \ldots, d_{i-1}(x), d_j(x) \pm c, d_{i+1}(x), \ldots, d_k(x) \rangle$$

- P ist $Q;R$: $g_P(x) = g_R(g_Q(x))$
- P ist LOOP x_i DO Q END:

$$\begin{aligned} h(0,x) &= x \\ h(n+1,x) &= g_Q(h(n,x)) \\ g_P(x) &= h(d_i(x),x) \end{aligned}$$

$g_P(x)$ berechnet den Zustand nach $d_i(x)$ Iterationen von Q.

Beweis (Forts.):

- P ist x_i := $x_j \pm c$:

$$g_P(x) = \langle d_0(x), \ldots, d_{i-1}(x), d_j(x) \pm c, d_{i+1}(x), \ldots, d_k(x) \rangle$$

- P ist $Q;R$: $g_P(x) = g_R(g_Q(x))$
- P ist LOOP x_i DO Q END:

$$\begin{aligned} h(0,x) &= x \\ h(n+1,x) &= g_Q(h(n,x)) \\ g_P(x) &= h(d_i(x),x) \end{aligned}$$

$g_P(x)$ berechnet den Zustand nach $d_i(x)$ Iterationen von Q.

Berechnet P die Funktion $f : \mathbb{N}^m \to \mathbb{N}$, dann ist f PR denn

$$f(x_1, \ldots, x_m) = d_0(g_P(\langle 0, x_1, \ldots x_m, \underbrace{0, \ldots, 0}_{k-m} \rangle)).$$

Beweis (Forts.): PR $\rightarrow$ LOOP
Sei $f : \mathbb{N}^k \rightarrow \mathbb{N}$ PR.

Beweis (Forts.): PR $\rightarrow$ LOOP
Sei $f : \mathbb{N}^k \rightarrow \mathbb{N}$ PR.
Mit Induktion über die Erzeugung von f konstruieren wir ein LOOP-Programm P_f, das f berechnet:

- 0:
  x_0 := 0

Beweis (Forts.): PR $\rightarrow$ LOOP
Sei $f : \mathbb{N}^k \rightarrow \mathbb{N}$ PR.
Mit Induktion über die Erzeugung von f konstruieren wir ein LOOP-Programm P_f, das f berechnet:

- 0:
  x_0 := 0
- Nachfolger-Funktion:
  x_0 := x_1 + 1

Beweis (Forts.): PR $\rightarrow$ LOOP
Sei $f : \mathbb{N}^k \rightarrow \mathbb{N}$ PR.
Mit Induktion über die Erzeugung von f konstruieren wir ein LOOP-Programm P_f, das f berechnet:

- 0:
  x_0 := 0
- Nachfolger-Funktion:
  x_0 := x_1 + 1
- Projektions-Funktionen, zB $\pi_2^3(x_1, x_2, x_3) = x_2$:
  x_0 := x_2

Beweis (Forts.): PR $\rightarrow$ LOOP

Sei $f : \mathbb{N}^k \rightarrow \mathbb{N}$ PR.
Mit Induktion über die Erzeugung von f konstruieren wir ein LOOP-Programm P_f, das f berechnet:

- 0:
  x_0 := 0
- Nachfolger-Funktion:
  x_0 := x_1 + 1
- Projektions-Funktionen, zB $\pi^3_2(x_1, x_2, x_3) = x_2$:
  x_0 := x_2
- Komposition, zB $f(x_1, x_2) = g(h_1(x_1, x_2), h_2(x_1, x_2))$:

Beweis (Forts.): PR $\rightarrow$ LOOP

Sei $f : \mathbb{N}^k \rightarrow \mathbb{N}$ PR.
Mit Induktion über die Erzeugung von f konstruieren wir ein LOOP-Programm P_f, das f berechnet:

- 0:
  x_0 := 0
- Nachfolger-Funktion:
  x_0 := x_1 + 1
- Projektions-Funktionen, zB $\pi^3_2(x_1, x_2, x_3) = x_2$:
  x_0 := x_2
- Komposition, zB $f(x_1, x_2) = g(h_1(x_1, x_2), h_2(x_1, x_2))$:

  P_{h_1}; x_{15} := x_0; P_{h_2}; x_1 := x_{15}; x_2 := x_0; P_g

Beweis (Forts.): PR $\rightarrow$ LOOP

Sei $f : \mathbb{N}^k \rightarrow \mathbb{N}$ PR.
Mit Induktion über die Erzeugung von f konstruieren wir ein LOOP-Programm P_f, das f berechnet:

- 0:
  x_0 := 0
- Nachfolger-Funktion:
  x_0 := x_1 + 1
- Projektions-Funktionen, zB $\pi^3_2(x_1, x_2, x_3) = x_2$:
  x_0 := x_2
- Komposition, zB $f(x_1, x_2) = g(h_1(x_1, x_2), h_2(x_1, x_2))$:

  P_{h_1}; x_{15} := x_0; P_{h_2}; x_1 := x_{15}; x_2 := x_0; P_g

  Funktions-Komposition wird simuliert durch Hintereinanderausführung (;) wobei Argumente und Zwischenergebnisse in separaten Variablen zwischengespeichert werden müssen.

Beweis (Forts.):

- Primitive Rekursion:

$$\begin{aligned} f(0, \overline{x}) &= g(\overline{x}) \\ f(y+1, \overline{x}) &= h(f(y, \overline{x}), y, \overline{x}) \end{aligned}$$

Beweis (Forts.):

- Primitive Rekursion:

$$\begin{aligned} f(0,\overline{x}) &= g(\overline{x}) \\ f(y+1,\overline{x}) &= h(f(y,\overline{x}),y,\overline{x}) \end{aligned}$$

Folgendes LOOP-Programm berechnet $f(y,\overline{x})$:

r := $g(\overline{x})$;
k := 0;
LOOP y DO
  r := $h(r,k,\overline{x})$
  k := $k+1$;
END
x_0 := r

Reversible Kodierung endlicher Zahlenfolgen als Zahlen:

Reversible Kodierung endlicher Zahlenfolgen als Zahlen:

$\{0,\ldots,k\}^*$ Kodiere $(i_1,\ldots,i_n)$ als Zahl $i_1\ldots i_n$ zur Basis k.

Reversible Kodierung endlicher Zahlenfolgen als Zahlen:

$\{0,\ldots,k\}^*$ Kodiere $(i_1,\ldots,i_n)$ als Zahl $i_1\ldots i_n$ zur Basis k.

$\mathbb{N}^n$ Mit iterierter Paarfunktion c: $\langle i_1,\ldots,i_n\rangle$
NB n muss beim Dekodieren bekannt sein denn zB $1 = c(0,1) = c(0,c(0,1))$.

$\mathbb{N}^*$ Auch mit $\langle\ldots\rangle$ reversibel kodierbar (Übung)

Reversible Kodierung endlicher Zahlenfolgen als Zahlen:

$\{0,\ldots,k\}^*$ Kodiere $(i_1,\ldots,i_n)$ als Zahl $i_1\ldots i_n$ zur Basis k.

$\mathbb{N}^n$ Mit iterierter Paarfunktion c: $\langle i_1,\ldots,i_n\rangle$
NB n muss beim Dekodieren bekannt sein denn zB $1 = c(0,1) = c(0,c(0,1))$.

$\mathbb{N}^*$ Auch mit $\langle\ldots\rangle$ reversibel kodierbar (Übung)

$\mathbb{N}^*$ Kodiere $(i_1,\ldots,i_n)$ als $2^{i_1}3^{i_2}\cdots p_n^{i_n}$
wobei p_n die n-te Primzahl ist.

Reversible Kodierung endlicher Zahlenfolgen als Zahlen:

$\{0,\ldots,k\}^*$ Kodiere $(i_1,\ldots,i_n)$ als Zahl $i_1\ldots i_n$ zur Basis k.

$\mathbb{N}^n$ Mit iterierter Paarfunktion c: $\langle i_1,\ldots,i_n\rangle$
NB n muss beim Dekodieren bekannt sein denn zB $1 = c(0,1) = c(0,c(0,1))$.

$\mathbb{N}^*$ Auch mit $\langle\ldots\rangle$ reversibel kodierbar (Übung)

$\mathbb{N}^*$ Kodiere $(i_1,\ldots,i_n)$ als $2^{i_1}3^{i_2}\cdots p_n^{i_n}$
wobei p_n die n-te Primzahl ist.
Dekodierung = Primzahlzerlegung

4.7 Die μ-rekursiven Funktionen

4.7 Die μ-rekursiven Funktionen

- Mit PR kann man nur beschränkt suchen: $n,\ldots,0$.

4.7 Die μ-rekursiven Funktionen

- Mit PR kann man nur beschränkt suchen: $n, \ldots, 0$.
- Die *unbschränkte* Suche $0, \ldots$ erfordert so etwas wie

$$f(n) \;=\; \ldots f(n+1) \ldots$$

4.7 Die μ-rekursiven Funktionen

- Mit PR kann man nur beschränkt suchen: $n, \ldots, 0$.
- Die *unbschränkte* Suche $0, \ldots$ erfordert so etwas wie

$$f(n) \;=\; \ldots f(n+1) \ldots$$

- Der μ-Operator formalisiert diese Art der Suche.
- Damit erhält man alle berechenbaren Funktionen.
- Andere Such- bzw Rekursionsschemata sind (im Prinzip!) nicht notwendig.

4.7 Die μ-rekursiven Funktionen

- Mit PR kann man nur beschränkt suchen: $n, \ldots, 0$.
- Die *unbschränkte* Suche $0, \ldots$ erfordert so etwas wie

$$f(n) \;=\; \ldots f(n+1) \ldots$$

- Der μ-Operator formalisiert diese Art der Suche.
- Damit erhält man alle berechenbaren Funktionen.
- Andere Such- bzw Rekursionsschemata sind (im Prinzip!) nicht notwendig.

Notation: $f(n) = \perp$ bedeutet „$f(n)$ ist undefiniert.“

Definition 4.42 (μ-Operator)

Sei $f : \mathbb{N}^{k+1} \to \mathbb{N}$ eine (nicht notwendigerweise totale) Funktion.

Definition 4.42 (μ-Operator)

Sei $f : \mathbb{N}^{k+1} \to \mathbb{N}$ eine (nicht notwendigerweise totale) Funktion.
Die durch Anwendung des μ-Operators entstehende Funktion
$\mu f : \mathbb{N}^k \to \mathbb{N}$ ist definiert durch:

Definition 4.42 (μ-Operator)

Sei $f : \mathbb{N}^{k+1} \to \mathbb{N}$ eine (nicht notwendigerweise totale) Funktion.
Die durch Anwendung des μ-Operators entstehende Funktion
$\mu f : \mathbb{N}^k \to \mathbb{N}$ ist definiert durch:

Intuitiv: $\mu f(\overline{x}) = find(0, \overline{x})$
$find(n, \overline{x}) = \textbf{if } f(n, \overline{x}) = 0 \textbf{ then } n \textbf{ else } find(n+1, \overline{x})$

Definition 4.42 (μ-Operator)

Sei $f : \mathbb{N}^{k+1} \to \mathbb{N}$ eine (nicht notwendigerweise totale) Funktion.
Die durch Anwendung des μ-Operators entstehende Funktion
$\mu f : \mathbb{N}^k \to \mathbb{N}$ ist definiert durch:

$$\overline{x} \mapsto \begin{cases} \min\{n \in \mathbb{N} \mid f(n, \overline{x}) = 0\} & \text{falls} \\ & \text{ein solches } n \text{ existiert und} \\ & f(m, \overline{x}) \neq \perp \text{ für alle } m \leq n \\ \perp & \text{sonst} \end{cases}$$

Intuitiv: $\mu f(\overline{x}) = find(0, \overline{x})$
$find(n, \overline{x}) = \textbf{if } f(n, \overline{x}) = 0 \textbf{ then } n \textbf{ else } find(n+1, \overline{x})$

Definition 4.42 (μ-Operator)

Sei $f : \mathbb{N}^{k+1} \to \mathbb{N}$ eine (nicht notwendigerweise totale) Funktion.
Die durch Anwendung des μ-Operators entstehende Funktion
$\mu f : \mathbb{N}^k \to \mathbb{N}$ ist definiert durch:

$$\overline{x} \mapsto \begin{cases} \min\{n \in \mathbb{N} \mid f(n, \overline{x}) = 0\} & \text{falls} \\ & \text{ein solches } n \text{ existiert und} \\ & f(m, \overline{x}) \neq \perp \text{ für alle } m \leq n \\ \perp & \text{sonst} \end{cases}$$

Intuitiv: $\mu f(\overline{x}) = find(0, \overline{x})$
$find(n, \overline{x}) = \textbf{if } f(n, \overline{x}) = 0 \textbf{ then } n \textbf{ else } find(n+1, \overline{x})$

Beispiel 4.43
Ist $f(n, x) = x \dot{-} (n + n)$
dann ist $\mu f(x) =$

Definition 4.42 (μ-Operator)

Sei $f : \mathbb{N}^{k+1} \to \mathbb{N}$ eine (nicht notwendigerweise totale) Funktion. Die durch Anwendung des μ-Operators entstehende Funktion $\mu f : \mathbb{N}^k \to \mathbb{N}$ ist definiert durch:

$$\overline{x} \mapsto \begin{cases} \min\{n \in \mathbb{N} \mid f(n, \overline{x}) = 0\} & \text{falls} \\ & \text{ein solches } n \text{ existiert und} \\ & f(m, \overline{x}) \neq \perp \text{ für alle } m \leq n \\ \perp & \text{sonst} \end{cases}$$

Intuitiv: $\mu f(\overline{x}) = find(0, \overline{x})$
$find(n, \overline{x}) = \textbf{if } f(n, \overline{x}) = 0 \textbf{ then } n \textbf{ else } find(n+1, \overline{x})$

Beispiel 4.43

Ist $f(n, x) = x \dot{-} (n + n)$
dann ist $\mu f(x) = \lceil x/2 \rceil$

Definition 4.44 (μR)

Die Menge der μ-rekursiven Funktionen ist die kleinste Teilmenge aller (nicht notwendigerweise totalen) Funktionen, die die Basisfunktionen $(0,\ +1,\ \pi_i^k)$ enthält

Definition 4.44 (μR)

Die Menge der μ-rekursiven Funktionen ist die kleinste Teilmenge aller (nicht notwendigerweise totalen) Funktionen, die die Basisfunktionen $(0,\ +1,\ \pi_i^k)$ enthält

und alle Funktionen, die man hieraus durch (evtl. wiederholte) Anwendung von Komposition, primitiver Rekursion oder des μ-Operators erzeugen kann.

Definition 4.44 (μR)

Die Menge der μ-rekursiven Funktionen ist die kleinste Teilmenge aller (nicht notwendigerweise totalen) Funktionen, die die Basisfunktionen $(0,\ +1,\ \pi_i^k)$ enthält und alle Funktionen, die man hieraus durch (evtl. wiederholte) Anwendung von Komposition, primitiver Rekursion oder des μ-Operators erzeugen kann.

Satz 4.45 (μR = WHILE)

Die μ-rekursiven sind genau die WHILE-berechenbaren Funktionen.

Definition 4.44 (μR)

Die Menge der μ-rekursiven Funktionen ist die kleinste Teilmenge aller (nicht notwendigerweise totalen) Funktionen, die die Basisfunktionen $(0,\ +1,\ \pi_i^k)$ enthält und alle Funktionen, die man hieraus durch (evtl. wiederholte) Anwendung von Komposition, primitiver Rekursion oder des μ-Operators erzeugen kann.

Satz 4.45 (μR = WHILE)

Die μ-rekursiven sind genau die WHILE-berechenbaren Funktionen.

Beweis: als Ergänzung des Beweises von PR = LOOP.

Definition 4.44 (μR)

Die Menge der μ-rekursiven Funktionen ist die kleinste Teilmenge aller (nicht notwendigerweise totalen) Funktionen, die die Basisfunktionen $(0,\ +1,\ \pi_i^k)$ enthält und alle Funktionen, die man hieraus durch (evtl. wiederholte) Anwendung von Komposition, primitiver Rekursion oder des μ-Operators erzeugen kann.

Satz 4.45 (μR = WHILE)

Die μ-rekursiven sind genau die WHILE-berechenbaren Funktionen.

Beweis: als Ergänzung des Beweises von PR = LOOP.

μR $\rightarrow$ WHILE:
Fall μf: Nach IA gibt es ein WHILE-Programm für f.

Definition 4.44 (μR)

Die Menge der μ-rekursiven Funktionen ist die kleinste Teilmenge aller (nicht notwendigerweise totalen) Funktionen, die die Basisfunktionen $(0,\ +1,\ \pi_i^k)$ enthält und alle Funktionen, die man hieraus durch (evtl. wiederholte) Anwendung von Komposition, primitiver Rekursion oder des μ-Operators erzeugen kann.

Satz 4.45 (μR = WHILE)

Die μ-rekursiven sind genau die WHILE-berechenbaren Funktionen.

Beweis: als Ergänzung des Beweises von PR = LOOP.
μR $\rightarrow$ WHILE:
Fall μf: Nach IA gibt es ein WHILE-Programm für f.

Dann ist μf auch WHILE-berechenbar:

x_0 `:=` 0`;` y `:=` $f(0,\overline{x})$`;`
`WHILE` $y \neq 0$ `DO` x_0 `:=` x_0 `+` 1`;` y `:=` $f(x_0,\overline{x})$ `END`

Beweis (Forts.):

WHILE $\rightarrow$ μR:
Fall P ist `WHILE` $x_i \neq 0$ `DO` Q `END`

Beweis (Forts.):

WHILE $\rightarrow \mu$R:
Fall P ist `WHILE` $x_i \neq 0$ `DO` Q `END`
Nach IA gibt es eine Funktion g_Q die Q berechnet.

Beweis (Forts.):

WHILE $\rightarrow \mu$R:
Fall P ist `WHILE` $x_i \neq 0$ `DO` Q `END`
Nach IA gibt es eine Funktion g_Q die Q berechnet.
Dann ist g_P wie folgt definierbar:

Beweis (Forts.):

WHILE $\rightarrow \mu$R:
Fall P ist `WHILE` $x_i \neq 0$ `DO` Q `END`
Nach IA gibt es eine Funktion g_Q die Q berechnet.
Dann ist g_P wie folgt definierbar:

$$\begin{aligned} h(0,x) &= x \\ h(n+1,x) &= g_Q(h(n,x)) \end{aligned}$$

Beweis (Forts.):

WHILE $\rightarrow \mu$R:
Fall P ist `WHILE` $x_i \neq 0$ `DO` Q `END`
Nach IA gibt es eine Funktion g_Q die Q berechnet.
Dann ist g_P wie folgt definierbar:

$$\begin{aligned} h(0,x) &= x \\ h(n+1,x) &= g_Q(h(n,x)) \\ h_i(n,x) &= d_i(h(n,x)) \end{aligned}$$

Beweis (Forts.):

WHILE $\rightarrow \mu$R:
Fall P ist `WHILE` $x_i \neq 0$ `DO` Q `END`
Nach IA gibt es eine Funktion g_Q die Q berechnet.
Dann ist g_P wie folgt definierbar:

$$\begin{aligned} h(0,x) &= x \\ h(n+1,x) &= g_Q(h(n,x)) \\ h_i(n,x) &= d_i(h(n,x)) \end{aligned}$$

$h_i(n,x)$ ist der Wert von x_i nach n Iterationen von Q.
Dh $\mu h_i(x)$ ist die Anzahl der Iterationen von Q bis $x_i = 0$,

Beweis (Forts.):

WHILE $\rightarrow \mu$R:
Fall P ist `WHILE` $x_i \neq 0$ `DO` Q `END`
Nach IA gibt es eine Funktion g_Q die Q berechnet.
Dann ist g_P wie folgt definierbar:

$$\begin{aligned} h(0,x) &= x \\ h(n+1,x) &= g_Q(h(n,x)) \\ h_i(n,x) &= d_i(h(n,x)) \end{aligned}$$

$h_i(n,x)$ ist der Wert von x_i nach n Iterationen von Q.
Dh $\mu h_i(x)$ ist die Anzahl der Iterationen von Q bis $x_i = 0$,
dh bis P terminiert.

$$g_P(x) = h(\mu h_i(x), x)$$

□

Beweis (Forts.):

WHILE $\rightarrow \mu$R:
Fall P ist `WHILE` $x_i \neq 0$ `DO` Q `END`
Nach IA gibt es eine Funktion g_Q die Q berechnet.
Dann ist g_P wie folgt definierbar:

$$\begin{aligned} h(0,x) &= x \\ h(n+1,x) &= g_Q(h(n,x)) \\ h_i(n,x) &= d_i(h(n,x)) \end{aligned}$$

$h_i(n,x)$ ist der Wert von x_i nach n Iterationen von Q.
Dh $\mu h_i(x)$ ist die Anzahl der Iterationen von Q bis $x_i = 0$,
dh bis P terminiert.

$$g_P(x) = h(\mu h_i(x), x)$$

□

Durch die Transformation

μR $\rightarrow$ WHILE mit einer Schleife (Kleene) $\rightarrow \mu$R

genügt also immer ein μ-Operator:

Durch die Transformation

μR $\rightarrow$ WHILE mit einer Schleife (Kleene) $\rightarrow$ μR

genügt also immer ein μ-Operator:

Durch die Transformation

μR $\rightarrow$ WHILE mit einer Schleife (Kleene) $\rightarrow$ μR

genügt also immer ein μ-Operator:

Korollar 4.46 (Kleene)

Für jede n-stellige μ-rekursive Funktionen f gibt es zwei $n+1$-stellige PR Funktionen h und h', so dass

$$f(\overline{x}) = h(\mu h'(\overline{x}), \overline{x})$$

4.8 Die Ackermann-Funktion

$$\begin{aligned} a(0, n) &= n+1 \\ a(m+1, 0) &= a(m, 1) \\ a(m+1, n+1) &= a(m, a(m+1, n)) \end{aligned}$$

4.8 Die Ackermann-Funktion

$$\begin{aligned} a(0, n) &= n+1 \\ a(m+1, 0) &= a(m, 1) \\ a(m+1, n+1) &= a(m, a(m+1, n)) \end{aligned}$$

- Dies ist keine PR Definition.

4.8 Die Ackermann-Funktion

$$\begin{aligned} a(\quad 0, \quad n) &= n+1 \\ a(m+1, \quad 0) &= a(m,1) \\ a(m+1, n+1) &= a(m, a(m+1, n)) \end{aligned}$$

- Dies ist keine PR Definition.
- Aber: $\not\Rightarrow$ a ist nicht PR

4.8 Die Ackermann-Funktion

$$\begin{aligned} a(\quad 0, \quad n) &= n+1 \\ a(m+1, \quad 0) &= a(m,1) \\ a(m+1, n+1) &= a(m, a(m+1, n)) \end{aligned}$$

- Dies ist keine PR Definition.
- Aber: $\not\Rightarrow$ a ist nicht PR

- Ziel: a ist berechenbar, total, aber nicht PR

4.8 Die Ackermann-Funktion

$$\begin{aligned} a(\quad 0, \quad n) &= n+1 \\ a(m+1, \quad 0) &= a(m,1) \\ a(m+1, n+1) &= a(m, a(m+1, n)) \end{aligned}$$

- Dies ist keine PR Definition.
- Aber: $\not\Rightarrow$ a ist nicht PR
- Ziel: a ist berechenbar, total, aber nicht PR

Fakt 4.47
Die Ackermann-Funktion ist (OCaml-)berechenbar.

Beispiel:

$$a(2,2) =$$
$$a(1, a(2, 1)) =$$
$$a(1, a(1, a(2, 0))) =$$
$$a(1, a(1, a(1, 1))) =$$
$$a(1, a(1, a(0, a(1, 0)))) =$$
$$a(1, a(1, a(0, a(0, 1)))) =$$
$$a(1, a(1, a(0, 2))) =$$
$$a(1, a(1, 3)) =$$
$$a(1, a(0, a(1, 2))) =$$
$$a(1, a(0, a(0, a(1, 1)))) =$$
$$a(1, a(0, a(0, a(0, a(1, 0))))) =$$
$$a(1, a(0, a(0, a(0, a(0, 1))))) =$$
$$a(1, a(0, a(0, a(0, 2)))) =$$
$$a(1, a(0, a(0, 3))) =$$
$$a(1, a(0, 4)) =$$
$$a(1, 5) =$$
$$a(0, a(1, 4)) =$$
$$a(0, a(0, a(1, 3))) =$$
$$a(0, a(0, a(0, a(1, 2)))) =$$
$$a(0, a(0, a(0, a(0, a(1, 1))))) =$$
$$a(0, a(0, a(0, a(0, a(0, a(1, 0)))))) =$$
$$a(0, a(0, a(0, a(0, a(0, a(0, 1)))))) =$$
$$a(0, a(0, a(0, a(0, a(0, 2))))) =$$
$$a(0, a(0, a(0, a(0, 3)))) =$$
$$a(0, a(0, a(0, 4))) =$$
$$a(0, a(0, 5)) =$$
$$a(0, 6) =$$
$$7$$

Beispiel:

$$
\begin{aligned}
&a(2,2) = \\
&a(1,a(2,1)) = \\
&a(1,a(1,a(2,0))) = \\
&a(1,a(1,a(1,1))) = \\
&a(1,a(1,a(0,a(1,0)))) = \\
&a(1,a(1,a(0,a(0,1)))) = \\
&a(1,a(1,a(0,2))) = \\
&a(1,a(1,3)) = \\
&a(1,a(0,a(1,2))) = \\
&a(1,a(0,a(0,a(1,1)))) = \\
&a(1,a(0,a(0,a(0,a(1,0))))) = \\
&a(1,a(0,a(0,a(0,a(0,1))))) = \\
&a(1,a(0,a(0,a(0,2)))) = \\
&a(1,a(0,a(0,3))) = \\
&a(1,a(0,4)) = \\
&a(1,5) = \\
&a(0,a(1,4)) = \\
&a(0,a(0,a(1,3))) = \\
&a(0,a(0,a(0,a(1,2)))) = \\
&a(0,a(0,a(0,a(0,a(1,1))))) = \\
&a(0,a(0,a(0,a(0,a(0,a(1,0)))))) = \\
&a(0,a(0,a(0,a(0,a(0,a(0,1)))))) = \\
&a(0,a(0,a(0,a(0,a(0,2))))) = \\
&a(0,a(0,a(0,a(0,3)))) = \\
&a(0,a(0,a(0,4))) = \\
&a(0,a(0,5)) = \\
&a(0,6) = \\
&7
\end{aligned}
$$

Scherzfrage: $a(6,6) = ?$

Beispiel:

$$
\begin{aligned}
&a(2,2) = \\
&a(1,a(2,1)) = \\
&a(1,a(1,a(2,0))) = \\
&a(1,a(1,a(1,1))) = \\
&a(1,a(1,a(0,a(1,0)))) = \\
&a(1,a(1,a(0,a(0,1)))) = \\
&a(1,a(1,a(0,2))) = \\
&a(1,a(1,3)) = \\
&a(1,a(0,a(1,2))) = \\
&a(1,a(0,a(0,a(1,1)))) = \\
&a(1,a(0,a(0,a(0,a(1,0))))) = \\
&a(1,a(0,a(0,a(0,a(0,1))))) = \\
&a(1,a(0,a(0,a(0,2)))) = \\
&a(1,a(0,a(0,3))) = \\
&a(1,a(0,4)) = \\
&a(1,5) = \\
&a(0,a(1,4)) = \\
&a(0,a(0,a(1,3))) = \\
&a(0,a(0,a(0,a(1,2)))) = \\
&a(0,a(0,a(0,a(0,a(1,1))))) = \\
&a(0,a(0,a(0,a(0,a(0,a(1,0)))))) = \\
&a(0,a(0,a(0,a(0,a(0,a(0,1)))))) = \\
&a(0,a(0,a(0,a(0,a(0,2))))) = \\
&a(0,a(0,a(0,a(0,3)))) = \\
&a(0,a(0,a(0,4))) = \\
&a(0,a(0,5)) = \\
&a(0,6) = \\
&7
\end{aligned}
$$

Scherzfrage: $a(6,6) = ?$

Der Beginn:

$$a(0,n) = n+1$$

Der Beginn:

$$
\begin{aligned}
a(0,n) &= n+1 \\
a(1,n) &= 2+(n+3)-3
\end{aligned}
$$

Der Beginn:

$$\begin{aligned} a(0,n) &= n+1 \\ a(1,n) &= 2+(n+3)-3 \\ a(2,n) &= 2(n+3)-3 \end{aligned}$$

Der Beginn:

$$\begin{aligned} a(0,n) &= n+1 \\ a(1,n) &= 2+(n+3)-3 \\ a(2,n) &= 2(n+3)-3 \\ a(3,n) &= 2^{n+3}-3 \end{aligned}$$

Der Beginn:

$$\begin{aligned} a(0,n) &= n+1 \\ a(1,n) &= 2+(n+3)-3 \\ a(2,n) &= 2(n+3)-3 \\ a(3,n) &= 2^{n+3}-3 \\ a(4,n) &= \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3}-3 \end{aligned}$$

Der Beginn:

$$\begin{aligned} a(0,n) &= n+1 \\ a(1,n) &= 2+(n+3)-3 \\ a(2,n) &= 2(n+3)-3 \\ a(3,n) &= 2^{n+3}-3 \\ a(4,n) &= \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3}-3 \\ a(5,n) &= ??? \end{aligned}$$

Mit Induktion über n:

$$a(m+1,n) = \underbrace{a(m,a(m,\ldots a(m}_{n+1},1)\ldots)$$

Der Beginn:

$$\begin{aligned} a(0,n) &= n+1 \\ a(1,n) &= 2+(n+3)-3 \\ a(2,n) &= 2(n+3)-3 \\ a(3,n) &= 2^{n+3}-3 \end{aligned}$$

Beispiel:

$a(2,2) =$
$a(1,a(2,1)) =$
$a(1,a(1,a(2,0))) =$
$a(1,a(1,a(1,1))) =$
$a(1,a(1,a(0,a(1,0)))) =$
$a(1,a(1,a(0,a(0,1)))) =$
$a(1,a(1,a(0,2))) =$
$a(1,a(1,3)) =$
$a(1,a(0,a(1,2))) =$
$a(1,a(0,a(0,a(1,1)))) =$
$a(1,a(0,a(0,a(0,a(1,0))))) =$
$a(1,a(0,a(0,a(0,a(0,1))))) =$
$a(1,a(0,a(0,a(0,2)))) =$
$a(1,a(0,a(0,3))) =$
$a(1,a(0,4)) =$
$a(1,5) =$
$a(0,a(1,4)) =$
$a(0,a(0,a(1,3))) =$
$a(0,a(0,a(0,a(1,2)))) =$
$a(0,a(0,a(0,a(0,a(1,1))))) =$
$a(0,a(0,a(0,a(0,a(0,a(1,0)))))) =$
$a(0,a(0,a(0,a(0,a(0,a(0,1)))))) =$
$a(0,a(0,a(0,a(0,a(0,2))))) =$
$a(0,a(0,a(0,a(0,3)))) =$
$a(0,a(0,a(0,4))) =$
$a(0,a(0,5)) =$
$a(0,6) =$
7

Ackermann als Familie von Funktionen A_m: $A_m(n) := a(m,n)$.

Ackermann als Familie von Funktionen A_m: $A_m(n) := a(m,n)$.
Nun gilt:

$$\begin{aligned} A_0(n) &= s(n) \\ A_{m+1}(n) &= A_m^{n+1}(1) \end{aligned}$$

Ackermann als Familie von Funktionen A_m: $A_m(n) := a(m,n)$.
Nun gilt:

$$\begin{aligned} A_0(n) &= s(n) \\ A_{m+1}(n) &= A_m^{n+1}(1) = \underbrace{A_m(\ldots A_m}_{n+1}(1)\ldots) \end{aligned}$$

Beobachtung: alle A_m sind total und PR

Ackermann als Familie von Funktionen A_m: $A_m(n) := a(m,n)$.
Nun gilt:

$$\begin{aligned} A_0(n) &= s(n) \\ A_{m+1}(n) &= A_m^{n+1}(1) = \underbrace{A_m(\ldots A_m}_{n+1}(1)\ldots) \end{aligned}$$

Beobachtung: alle A_m sind total und PR (mit Ind. über m)

Mit Currying (z.B. in Haskell):

```
a 0      n  =  n+1
a (m+1)  n  =  iter (n+1) (a m) 1

iter 0      f x  =
iter (n+1)  f x  =
```

Ackermann als Familie von Funktionen A_m: $A_m(n) := a(m,n)$.
Nun gilt:

$$\begin{aligned} A_0(n) &= s(n) \\ A_{m+1}(n) &= A_m^{n+1}(1) = \underbrace{A_m(\ldots A_m}_{n+1}(1)\ldots) \end{aligned}$$

Beobachtung: alle A_m sind total und PR (mit Ind. über m)

Mit Currying (z.B. in Haskell):

```
a 0      n  =  n+1
a (m+1)  n  =  iter (n+1) (a m) 1
```

Man kann auch direkt zeigen:

Lemma 4.48
Die Ackermann-Funktion ist total.

Man kann auch direkt zeigen:

Lemma 4.48
Die Ackermann-Funktion ist total.

Beweis:
Mit Induktion über m: $\forall n \in \mathbb{N}.\ a(m,n) \neq \bot$ (1)