Primitiv rekursive und μ-rekursive Funktionen

Hans U. Simon (RUB)

Email: simon@lmi.rub.de

Homepage: http://www.ruhr-uni-bochum.de/lmi

Primitiv rekursive Funktionen

Basisfunktionen:

konstante Funktionen:	$c(n_1, \ldots, n_k) = c$
Projektionen:	$\pi_i^k(n_1, \ldots, n_k) = n_i$
Nachfolgerfunktion:	$s(n) = n + 1$

Beachte: $\pi_1^1(n) = n$ ist die identische Funktion.

Die Basisfunktionen sind primitiv rekursiv sowie alle Funktionen, die sich induktiv wie folgt ergeben:

Einsetzungsschema Jede Funktion, die durch „Komposition“ von primitiv rekursiven Funktionen entsteht, ist primitiv rekursiv.

primitives Rekursionsschema Jede Funktion, die sich durch „primitive Rekursion (Induktion)“ aus primitiv rekursiven Funktionen ergibt, ist primitiv rekursiv.

Komposition und primitive Rekursion

Das Einsetzungsschema:

Gegeben seien primitiv rekursive Funktionen $h : \mathbb{N}^r \to \mathbb{N}$ und $g_i : \mathbb{N}^k \to \mathbb{N}$ für $i = 1, \ldots, r$. Dann ist auch die Funktion $f : \mathbb{N}^k \to \mathbb{N}$ mit

$$f(x) = h\left(g_1(x), \ldots, g_r(x)\right)$$

primitiv rekursiv.

Das Schema der primitiven Rekursion:

Gegeben seien primitiv rekursive Funktionen $g : \mathbb{N}^k \to \mathbb{N}$ und $h : \mathbb{N}^{k+2} \to \mathbb{N}$. Dann ist die folgende (induktiv definierte) Funktion $f : \mathbb{N}^{k+1} \to \mathbb{N}$ primitiv rekursiv:

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

Vertauschen, Identifikation und Konstantsetzung von Variablen

In Verbindung mit den Projektionsabbildungen π_i^k und den konstanten Funktionen kann das Einsetzungsschema benutzt werden, um Variablen zu vertauschen, zu identifizieren oder konstant zu setzen, ohne die Klasse der primitiv rekursiven Funktionen zu verlassen.

Beispiel: Wenn $f(u, v, w, x, y)$ primitiv rekursiv ist, dann ist auch

$$\begin{aligned} g(a, b, c) &= f(b, b, c, a, 1) \\ &= f\left(\pi_2^3(a, b, c), \pi_2^3(a, b, c), \pi_3^3(a, b, c), \pi_1^3(a, b, c), 1\right) \end{aligned}$$

primitiv rekursiv.

Addition und Multiplikation sind primitiv rekursiv

Addition Nutze aus, dass $(n+1)+x = (n+x)+1$:

$$\begin{aligned} \mathrm{add}(0,x) &= x \\ \mathrm{add}(n+1,x) &= s(\mathrm{add}(n,x)) \end{aligned}$$

Dies entspricht dem primitiven Rekursionsschema mit $g = \pi_1^1$ (identische Funktion) und $h = s \circ \pi_1^3$ wegen

$$s(\mathrm{add}(n,x)) = s(\pi_1^3(\mathrm{add}(n,x),n,x)) \ .$$

Ab jetzt geben wir g und h bei Verwendung des primitiven Rekursionsschema nicht immer explizit an.

Multiplikation Nutze aus, dass $(n+1)x = nx + x$:

$$\begin{aligned} \mathrm{mult}(0,x) &= 0 \\ \mathrm{mult}(n+1,x) &= \mathrm{add}(\mathrm{mult}(n,x),x) \end{aligned}$$

Modifizierte Differenz ist primitiv rekursiv

Funktion $\mathrm{sub}(x, y)$ soll die modifizierte Differenz

$$x \dot{-} y = \max\{0, x - y\}$$

darstellen. Funktion

$$u(n) = \max\{0, n - 1\}$$

ist entsprechend die modifizierte Vorgängerfunktion. Wegen

$$\begin{aligned} u(0) &= 0 \\ u(n+1) &= n \end{aligned}$$

ist u primitiv rekursiv. Wegen

$$\begin{aligned} \mathrm{sub}(x, 0) &= x \\ \mathrm{sub}(x, y+1) &= u(\mathrm{sub}(x, y)) \end{aligned}$$

ist dann auch sub primitiv rekursiv.

Exkurs: Bijektion zwischen $\mathbb{N}^2$ und $\mathbb{N}$

Folgende Funktion $c(x, y)$, unten angegeben als Matrix, liefert eine (bijektive) Abzählung aller Paare $(x, y) \in \mathbb{N} \times \mathbb{N}$:

0	2	5	9	14	$\cdots$
1	4	8	13	19	$\cdots$
3	7	12	18	25	$\cdots$
6	11	17	24	32	$\cdots$
10	16	23	31	40	$\cdots$

$\cdots$

Nach diesem Schema gilt (Denksportaufgabe!)

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

Primitive rekursive Zahlkodierung von Tupeln

Wegen

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

ist die Funktion $\binom{n}{2}$ primitiv rekursiv. Mit dem Einsetzungsschema folgt dann leicht, dass erstens

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

und zweitens

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

primitiv rekursiv ist. Abbildung $\langle \cdots \rangle$ bettet $\mathbb{N}^{k+1}$ (injektiv) in $\mathbb{N}$ ein. (Kodierung eines Zahlentupels durch **eine** Zahl).

Rückgewinnung des Zahlentupels

Wie können wir aus der Zahl $n = \langle n_0, n_1, \ldots, n_k \rangle$ das Tupel $(n_0, n_1, \ldots, n_k)$ zurückgewinnen (mathematisch die Frage nach der Umkehrfunktion)?

Als bijektive Abbildung hat $c(x, y)$ eine Umkehrfunktion (e, f) mit

$$e(c(x, y)) = x \text{ und } f(c(x, y)) = y \ .$$

Wegen

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

ergibt sich

$$\begin{aligned}
d_0(n) &:= e(n) &= n_0 \\
d_1(n) &:= e(f(n)) &= n_1 \\
&\cdots \\
d_k(n) &:= e(\underbrace{f(f(\cdots f}_{k-mal}(n) \cdots)) &= n_k
\end{aligned}$$

Primitive Rekursivität der Dekodierung

Hilfssatz: Funktionen e, f mit

$$e(c(x,y)) = x \text{ und } f(c(x,y)) = y$$

sind primitiv rekursiv.
Den etwas kniffligen Beweis verschieben wir auf später (oder lassen ihn evtl. aus.)

Mit dem Einsetzungsschema ergibt sich dann sofort die
Folgerung Funktionen $d_0, d_1, \ldots, d_k$ mit

$$d_i(\langle n_0, \ldots, n_k \rangle) = n_i$$

für $i = 0, 1 \ldots, k$ sind primitiv rekursiv.

Primitive Rekursivität LOOP–berechenbarer Funktionen

Wir betrachten ein LOOP-Programm P, das eine Teilmenge der Variablen $x_0, x_1, \ldots, x_k$ verwendet. Das Verhalten von P ist vollständig beschrieben durch die Funktion $g_P : \mathbb{N} \rightarrow \mathbb{N}$ mit:

$$g_P(\langle \overbrace{a_0, a_1, \ldots, a_k}^{Anfangsbelegung} \rangle) = \langle \overbrace{b_0, b_1, \ldots, b_k}^{Endbelegung} \rangle \ .$$

Satz Funktion g_P zu einem LOOP-Programm P ist primitiv rekursiv.

Folgerung Jede LOOP-berechenbare Funktion ist primitiv rekursiv.

Denn: Wenn P mit Variablen $x_0, x_1, \ldots, x_k$ die Funktion $f : \mathbb{N}^r \rightarrow \mathbb{N}$ berechnet, dann gilt

$$f(n_1, \ldots, n_r) = d_0(g_P(\langle 0, n_1, \ldots, n_r, \underbrace{0, \ldots, 0}_{(k-r)-mal} \rangle)) \ .$$

Primitive Rekursivität von g_P

Q durchlaufe die „Teilprogramme“ von P, beginnend bei einfachen Wertzuweisungen und fortschreitend zu zunehmend komplexeren Teilprogrammen. Das zuletzt durchlaufene „Teilprogramm“ ist P selbst.

zu zeigen: Für jedes Teilprogramm Q ist g_Q eine primitiv rekursive Funktion.

Fall 1 Q hat die Form $x_i := x_j \pm c$.
Dann gilt $g_Q(\langle a_0, a_1, \ldots, a_k\rangle) = \langle b_0, b_1, \ldots, b_k\rangle$ mit

$$b_l = \begin{cases} a_l & \text{falls } l \neq i \\ a_j \pm c & \text{falls } l = i \end{cases} .$$

Wegen $a_l = d_l(\langle a_0, a_1, \ldots, a_k\rangle)$ folgt die primitive Rekursivität von g_Q unmittelbar aus dem Einsetzungsschema.

Komplexere Teilprogramme

Fall 2 Q hat die Form „$Q'; Q''$“

Da dann Q', Q'' zuvor schon betrachtet wurden ist die primitive Rekursivität von $g_{Q'}$ und $g_{Q''}$ schon geklärt. Wegen $g_Q = g_{Q''} \circ g_{Q'}$ ist nach dem Einsetzungsschema auch g_Q primitiv rekursiv.

Komplexere Teilprogramme (fortgesetzt)

Fall 3 Q hat die Form „LOOP x_i DO Q' END“.
Da dann Q' zuvor schon betrachtet wurde, ist die primitive Rekursivität von $g_{Q'}$ bereits geklärt. Betrachte Hilfsfunktion

$$h(n,x) = \underbrace{g_{Q'}(g_{Q'}(\cdots(g_{Q'}}_{n-mal}(x)\cdots) \ .$$

Mit dem Schema der primitiven Rekursion

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

folgt die primitive Rekursivität von h. Die primitive Rekursivität von g_Q ergibt sich dann mit dem Einsetzungsschema aus

$$g_Q(\langle a_0, a_1, \ldots, a_k \rangle) = h(a_i, \langle a_0, a_1, \ldots, a_k \rangle) \ .$$

LOOP-Berechenbarkeit von primitiv rekursiven Funktionen

Induktion über den Aufbau von primitiv rekursiven Funktionen:

1. Die Basisfunktionen (Konstanten, Projektionen, Nachfolgerfunktion) sind offensichtlich LOOP-berechenbar.

2. Betrachte eine Funktion f der Form

$$f(x) = h(g_1(x), \ldots, g_r(x)) \ ,$$

wobei gemäß Induktionsvoraussetzung $h, g_1, \ldots, g_r$ LOOP-berechenbar sind. Dann kann ein LOOP-Programm $f(x)$ nach folgendem Schema berechnen:

$$y_1 := g_1(x) \ ; \ \cdots \ ; \ y_r := g_r(x) \ ; \ x_0 := h(y_1, \ldots, y_r)$$

LOOP-Berechenbarkeit von primitiv rekursiven Funktionen (fortgesetzt)

3. Betrachte eine Funktion f der Form

$$f(0,x) = g(x) \text{ und } f(n+1,x) = h(f(n,x),n,x) \;\; ,$$

wobei gemäß Induktionsvoraussetzung g und h LOOP-berechenbar sind. Dann kann ein LOOP-Programm $f(n,x)$ nach folgendem Schema berechnen:

$$z := 0 \; ; \; x_0 := g(x) \; ; \; \text{LOOP } x_1 \text{ DO } z := z+1 \; ; \; x_0 := h(x_0,z,x) \text{ END}$$

Kommentare:

– Variable x_1 enthält den Eingabeparameter n.

– Nach i Iterationen hat x_0 den Wert $f(i,x)$.

Hauptresultate

Es ergibt sich der

Satz: Die Klasse der primitiv rekursiven Funktionen stimmt mit der Klasse LOOP-berechenbaren Funktionen über ein.

Wir werden die Klasse der primitiv rekursiven Funktionen durch Einführung des sogenannten μ-Operators zur Klasse der μ-rekursiven Funktionen erweitern.

Ziel: Die Klasse der μ-rekursiven Funktionen stimmt mit der Klasse WHILE-berechenbaren Funktionen über ein.

Der μ-Operator

Für eine gegebene (evtl. partielle) Funktion $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ bezeichne $\mu f : \mathbb{N}^k \rightarrow \mathbb{N}$ die Funktion

$$\mu f(x) := \min\{n|\ f(n,x) = 0, \forall m < n : f(m,x) \text{ ist definiert}\}$$

verbunden mit der Konvention $\min \emptyset =$„undefiniert“.

Intuitive Bemerkung Falls $\mu f(x)$ definiert ist, dann liefert diese Funktion eine Art „kleinste Nullstelle“ für die Funktion $f(n,x)$ (aufgefasst als Funktion in n).

Definition Die Klasse der μ-rekursiven Funktionen ist die kleinste Klasse von (evtl. partiellen) Funktionen, die die Basisfunktionen enthält und abgeschlossen ist unter Einsetzung, primitiver Rekursion und der Anwendung des μ-Operators.

μ–Rekursivität WHILE–berechenbarer Funktionen

Wir betrachten ein WHILE-Programm P, das eine Teilmenge der Variablen $x_0, x_1, \ldots, x_k$ verwendet. Das Verhalten von P ist vollständig beschrieben durch die (evtl. partielle) Funktion $g_P : \mathbb{N} \rightarrow \mathbb{N}$ mit:

$$g_P(\langle \overbrace{a_0, a_1, \ldots, a_k}^{Anfangsbelegung} \rangle) = \langle \overbrace{b_0, b_1, \ldots, b_k}^{Endbelegung} \rangle$$

verbunden mit der Konvention, dass g_P undefiniert ist, wenn P nicht terminiert.

Satz Funktion g_P zu einem WHILE-Programm P ist μ-rekursiv.

Folgerung Jede WHILE-berechenbare Funktion ist μ-rekursiv.

μ-Rekursivität von g_P

Q durchlaufe die „Teilprogramme“ von P, beginnend bei einfachen Wertzuweisungen und fortschreitend zu zunehmend komplexeren Teilprogrammen. Das zuletzt durchlaufene „Teilprogramm“ ist P selbst.

zu zeigen: Für jedes Teilprogramm Q ist g_Q eine μ rekursive Funktion.

Die Fälle der Wertzuweisung und der Komposition zweier WHILE-Programme lassen sich abhandeln wie bei der analogen Überlegung für LOOP-Programme. Wesentlich neu ist nur der Fall der WHILE-Anweisung.

WHILE-Anweisung und μ-Operator

Neuer Fall Q hat die Form „WHILE $x_i \neq 0$ DO Q' END“.
Da dann Q' zuvor schon betrachtet wurde, ist die μ-Rekursivität von $g_{Q'}$ bereits geklärt. Betrachte die μ-rekursive (!) Hilfsfunktion

$$h(n,x) = \underbrace{g_{Q'}(g_{Q'}(\cdots(g_{Q'}}_{n-mal}(x)\cdots) \ .$$

Beachte: $d_i(h(n,x))$ ist der Wert der Variablen x_i nach n Ausführungen von Q. Die WHILE-Anweisung führt daher Q insgesamt

$$\min\{n|\ d_i(h(n,x)) = 0\} = \mu(d_i \circ h)(x)$$

mal aus (sofern sie terminiert). Mit

$$g_Q(x) = h\left(\mu(d_i \circ h)(x), x\right)$$

ergibt sich die μ-Rekursivität von g_Q.

WHILE-Berechenbarkeit von μ-rekursiven Funktionen

Induktion über den Aufbau von μ-rekursiven Funktionen: Wegen der Analogie zur LOOP-Berechenbarkeit von primitiv rekursiven Funktionen beschränken wir uns auf den

Neuen Fall: Betrachte eine (evtl. partielle) Funktion der Form

$$\mu f(x) = \min\{n |\ f(n,x) = 0, \forall m < n : f(m,x) \text{ ist definiert}\} \ .$$

Gemäß Induktionsvoraussetzung ist $f(n,x)$ WHILE-berechenbar. Dann können wir $\mu f(x)$ nach folgendem Schema berechnen:

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

Kommentar: Sofern $\mu f(x)$ definiert ist, enthält Variable x_0 am Ende die kleinste Zahl $n \in \mathbb{N}$ mit $f(n,x) = 0$ (kleinste Nullstelle).