

# Einige primitiv-rekursive Funktionen

Sebastian Wild und Markus E. Nebel

13. März 2012

## Inhaltsverzeichnis

1	Notationen	1
2	Die Identitätsfunktion	2
3	Beliebig-stellige konstante Funktionen	2
4	Endlich-quantisierte Prädikate	3
5	Die mysteriöse $\pi$ -Dezimalstellen-Funktion	4
6	Exponent der $i$ -ten Primzahl	5
7	Verschränkt rekursive Funktionen	6

---

## 1 Notationen

Schreibe  $f \in \mathcal{P}_n$  genau dann, wenn  $f$  eine **n-stellige**, primitiv-rekursive Funktion ist.

Verwende  $\vec{x} = x_1, \dots, x_n$ , falls die Dimension  $n$  aus dem Zusammenhang klar ist.

Für geläufige Schreibweisen  $S$  sei  $f_S$  die zugehörige Funktion, also z. B. zum Infix-Additions-Operator  $+$  ist  $f_+$  die zweistellige Funktion  $f_+(a, b) = a + b$ .

(Eingeklammerte Teile – wie dieser – dienen hauptsächlich dem Verständnis. Wenn nicht ausdrücklich gefordert, wird üblicherweise nicht erwartet, dass entsprechende Kommentare durch den Studierenden bei der Lösung entsprechender Aufgaben gemacht werden.)

## 2 Die Identitätsfunktion

Behauptung: Die einstellige Identitätsfunktion  $id(x) := x$  für alle  $x \in \mathbb{N}_0$  ist primitiv-rekursiv.

Beweis: Definiere

$$\begin{aligned} i(0) &= g() := C^0 \\ i(x+1) &= h(\underbrace{i(x)}_z, x) := s(z) \end{aligned}$$

$g$  und  $h$  sind elementare Funktionen der primitiv-rekursiven Funktionen und  $i$  ist nach dem Schema der *primitiven Rekursion* definiert.  $\hookrightarrow i \in \mathcal{P}_1$ .

$$\left( \begin{array}{l} \text{Bleibt noch zu zeigen, dass } i = id. \text{ Dazu führen wir eine Induktion über } x. \\ \text{Induktionsanfang } x = 0: \quad i(0) = C^0 = 0 = id(0) \\ \text{Induktionsschritt } x \rightarrow x+1: \quad \underset{\text{def}}{i(x+1)} = \underset{\text{def}}{s(i(x))} = i(x) + 1 \underset{\text{IV}}{=} id(x) + 1 = x + 1 = id(x+1) \quad \square \\ \text{Alternativ kann man argumentieren: Nach Definition ist für jedes } m \in \mathbb{N} \text{ die Projektion } U_m^1 \\ \text{eine elementare primitiv-rekursive Funktion; für } m = 1 \text{ ist } U_1^1 = id. \end{array} \right)$$

## 3 Beliebige-stellige konstante Funktionen

Behauptung: Für beliebiges  $n \in \mathbb{N}$  und ein festes  $k \in \mathbb{N}_0$  ist  $C_n^k(x_1, \dots, x_n) := k$  für alle  $x_1, \dots, x_n \in (\mathbb{N}_0)^n$  primitiv-rekursiv.

Beweis:  $C_n^k(x_1, \dots, x_n) = C_1^k(U_n^1(x_1, \dots, x_n)) = C_1^k(x_1)$ . Das bedeutet, es genügt, die einstelligen konstanten Funktionen zu betrachten. Definiere dazu

$$\begin{aligned} c^k(0) &= g() := s(\underbrace{\dots s(s(C^0)) \dots}_{k \text{ mal}}) \\ c^k(x+1) &= h(\underbrace{c^k(x)}_z, x) := z \end{aligned}$$

Für ein konkretes  $k$  ist  $g$  eine konkrete primitiv-rekursive Funktion (siehe auch Buch, Seite 122). Offensichtlich ist  $h = U_2^1$  primitiv-rekursiv und damit auch  $c^k$ , da sie nach dem Schema der primitiven Rekursion definiert ist.

$$\left( \begin{array}{l} \text{Eine triviale Induktion wie oben bestätigt } C_1^k = c^k: \\ \text{IA } x = 0: \quad c^k(0) = s(\underbrace{\dots s(s(C^0)) \dots}_{k \text{ mal}}) = k = C_1^k(0) \\ \text{IS } x \rightarrow x+1: \quad c^k(x+1) = \underset{\text{IV}}{c^k(x)} = C_1^k(x) = k = C_1^k(x+1) \quad \square \end{array} \right)$$

## 4 Endlich-quantisierte Prädikate

Behauptung: Die Funktionen  $f_{[\forall x \leq c P(\vec{y}, x)]}$  und  $f_{[\exists x \leq c P(\vec{y}, x)]}$  sind für beliebige primitiv-rekursive Prädikate  $P \in \mathcal{P}_{n+1}$  [ $n \geq 0$ ] selbst primitiv-rekursiv.

Beachte: Sowohl  $\forall x \leq c P(\vec{y}, x)$ , als auch  $\exists x \leq c P(\vec{y}, x)$  sind *endliche* Quantisierungen des Prädikats  $P$ . Da  $x$  eine gebundene Variable ist, sind beide quantisierten Prädikate wieder  $(n+1)$ -stellige Funktionen/Prädikate! Das heißt, die Behauptung lautet formal:

$$P \in \mathcal{P}_{n+1} \leadsto \left( f_{[\forall x \leq c P(\vec{y}, x)]} \in \mathcal{P}_{n+1} \wedge f_{[\exists x \leq c P(\vec{y}, x)]} \in \mathcal{P}_{n+1} \right).$$

Beweis: Nach Satz 4.3 aus dem Buch sind die primitiv-rekursiven Prädikate unter den binären Verknüpfungen  $\wedge$  und  $\vee$  sowie unter dem unären Operator  $\neg$  abgeschlossen, d. h.

$$P \in \mathcal{P}_n, Q \in \mathcal{P}^m \leadsto f_{[P \wedge Q]}, f_{[P \vee Q]} \in \mathcal{P}_{n+m} \wedge f_{[\neg P]} \in \mathcal{P}_n.$$

Sei nun  $P \in \mathcal{P}_{n+1}$  wie oben in der Behauptung gefordert. Definiere

$$\begin{aligned} A(\vec{y}, 0) &= g(\vec{y}) := P(\vec{y}, 0), \\ A(\vec{y}, c+1) &= h_1 \left( \underbrace{A(\vec{y}, c)}_z, \vec{y}, c \right) := z \wedge P(\vec{y}, c+1) \end{aligned}$$

und

$$\begin{aligned} E(\vec{y}, 0) &= g(\vec{y}) := P(\vec{y}, 0), \\ E(\vec{y}, c+1) &= h_2 \left( \underbrace{E(\vec{y}, c)}_z, \vec{y}, c \right) := z \vee P(\vec{y}, c+1). \end{aligned}$$

Hilfsbehauptung: Die Funktionen  $g$ ,  $h_1$  und  $h_2$  sind primitiv-rekursiv.

Beweis: Wir geben für alle drei Funktionen eine Konstruktion aus primitiv-rekursiven Funktionen an:

- $g(\vec{y}) = P(U_n^1(\vec{y}), \dots, U_n^n(\vec{y}), C_n^0(\vec{y}))$  [durch Einsetzen]
- $h_1(z, \vec{y}, c) = f_{\wedge}(U_{n+2}^1(z, \vec{y}, c), j(z, \vec{y}, c))$  mit  $j(\vec{x}) = P(U_{n+2}^2(\vec{x}), \dots, U_{n+2}^{n+1}(\vec{x}), s(U_{n+2}^{n+2}(\vec{x})))$
- $h_2(z, \vec{y}, c) = f_{\vee}(U_{n+2}^1(z, \vec{y}, c), j(z, \vec{y}, c))$

Da  $A$  und  $E$  durch primitive Rekursion aus diesen Funktionen entstehen, sind sie selbst auch in  $\mathcal{P}$ .

$$\left( \begin{array}{l} \text{Korrektheit: Aus der Logik sind die folgenden Äquivalenzen bekannt [zur Erinnerung:} \\ \text{es sind } \textit{endliche} \text{ Quantorisierungen!]} \\ \\ \forall x \leq c \, p(x) \iff p(0) \wedge p(1) \wedge \dots \wedge p(c) \\ \exists x \leq c \, p(x) \iff p(0) \vee p(1) \vee \dots \vee p(c) \\ \\ \text{IA } c = 0: \quad A(\vec{y}, 0) = P(\vec{y}, 0) = \forall x \leq 0 \, P(\vec{y}, 0) \text{ und } E(\vec{y}, 0) = P(\vec{y}, 0) = \exists x \leq 0 \, P(\vec{y}, 0). \text{ Man} \\ \text{behalte in Erinnerung, dass unser Universum (für } x) \text{ stets } \mathbb{N}_0 \text{ ist.} \\ \text{IS } c \rightarrow c + 1: \quad A(\vec{y}, c + 1) = A(\vec{y}, c) \wedge P(\vec{y}, c + 1) \stackrel{\text{IV}}{=} [\forall x \leq c \, P(\vec{y}, x)] \wedge P(\vec{y}, c + 1) \stackrel{(1)}{=} \\ \forall x \leq c + 1 \, P(\vec{y}, x) \text{ und analog für } E. \quad \square \end{array} \right) \quad (1)$$

## 5 Die mysteriöse $\pi$ -Dezimalstellen-Funktion

Behauptung: Die Funktion  $h$  mit

$$h(n) := \begin{cases} 1 & \text{falls die Dezimaldarstellung von } \pi \text{ eine Folge} \\ & \text{von } n \text{ aufeinanderfolgenden Nullen enthält} \\ 0 & \text{sonst} \end{cases}$$

ist [sogar<sup>1</sup>] primitiv-rekursiv berechenbar.

Beweis: Gleich vorneweg: Wir sind nicht in der Lage, eine solche Funktion anzugeben. Vielleicht ist das niemand. Das Schöne: Das muss man auch nicht!

Trivialerweise gilt für beliebiges  $n \in \mathbb{N}$ :  $h(n) = 1 \iff \forall m \leq n \, h(m) = 1$ , denn jede Folge von  $n$  Nullen enthält auch eine Folge von  $m \leq n$  Nullen. Nun ist uns zwar die tatsächliche, *vollständige* Dezimaldarstellung von  $\pi$  nicht bekannt, doch es gilt sicherlich einer der beiden Fälle:

- (1) Es existiert ein  $N \in \mathbb{N}_0$ , das die Länge der längsten Folge von Nullen in der Dezimaldarstellung von  $\pi$  ist; also  $h(N) = 1$ , aber  $h(N + 1) = 0$ . Nach obiger Überlegung ist dann  $\forall n \, h(n) = f_{\leq}(n, N) = f_{\leq}^N(id(n), C_1^N(n))$ . Diese Funktion ist aber bekanntermaßen primitiv-rekursiv (nach Buch), und zwar für jedes  $N \in \mathbb{N}_0$ .

In diesem Fall ist also  $h \in \mathcal{P}$ .

- (2) Es existiert *kein*  $N \in \mathbb{N}_0$ , das die Länge der längsten Folge von Nullen in der Dezimaldarstellung von  $\pi$  ist, d. h. man kann Folgen von Nullen *beliebiger* Länge finden. Dann ist sicherlich  $\forall n \, h(n) = 1$  und damit  $h = C_1^1 \in \mathcal{P}$ .

Weitere Fälle kann es nicht geben, egal wie  $\pi$  aussieht! Also ist  $h$  mit Sicherheit primitiv-rekursiv: Wir wissen zwar nicht, wie  $h$  aussieht, aber alle der [unendlich vielen!] möglichen Kandidaten sind selbst in  $\mathcal{P}$ , womit auch  $h$  nichts anderes übrigbleibt als primitiv-rekursiv zu sein.  $\square$

---

<sup>1</sup>und damit natürlich auch  $\mu$ -rekursiv.

6 Exponent der  $i$ -ten Primzahl

Behauptung:  $(x)_i \in \mathcal{P}_1$

Beweis: Verwende die Funktion  $p$  aus dem Buch mit  $p(i) = i$ -te Primzahl. Diese ist – wie dort gezeigt – primitiv-rekursiv.

Wir schließen den unschönen Spezialfall  $i = 0$ , d. h.  $p(i) = 1$  vorerst aus.

Dann ist der Exponent der  $i$ -ten Primzahl  $p(i)$  in der Primfaktorzerlegung von  $x$  [offensichtlicherweise] die größte Potenz  $(p(i))^z$ , die  $x$  teilt, d. h.  $\max \{z : (p(i))^z \mid x\}$ . Nun gilt außerdem

$$\forall j \ (p(i))^{j-1} \mid (p(i))^j,$$

d. h.  $\{z : (p(i))^z \mid x\} = \{0, 1, 2, \dots, (x)_i\}$ . Das ist wichtig, weil unsere Funktion später anschaulich  $z$  erhöht, bis die  $z$ -te Potenz *kein* Teiler von  $x$  mehr ist.

Für die primitiv-rekursiven Funktionen müssen wir für  $z$  nun eine obere Schranke finden.<sup>2</sup> Mit den Einschränkungen oben [also  $i \neq 0$ ] gilt  $\forall x \in \mathbb{N}_0, i > 0 \ (p(i))^{x+1} > x$ .

$$\left( \begin{array}{l} p(i) \text{ ist hier immer } \geq 2, \text{ und für } 2 \text{ gilt } \forall x \geq 0 \ [2^{x+1} > x]. \text{ Kleine Induktion dazu:} \\ \text{IA } 2^{0+1} = 2 > 0 \\ \text{IS } 2^{(x+1)+1} = 2 \cdot 2^{x+1} \underset{\text{IV}}{>} 2 \cdot x = x + x \geq x + 1 \\ \text{Außerdem ist offensichtlich nach Definition } p(i+1) > p(i), \text{ woraus wegen } x \geq 0 \text{ sofort} \\ (p(i+1))^{x+1} > (p(i))^{x+1} > x \text{ folgt und damit die Behauptung } \forall x \geq 0, i > 0 \ (p(i))^{x+1} > x. \end{array} \right)$$

Das bedeutet  $(x)_i < x + 1$  für alle  $i > 0$ , denn sonst wäre  $(p(i))^{(x)_i} > x$  und damit sicher *kein* Teiler von  $x$ .

Damit können wir für  $i > 0$  folgende primitiv-rekursive Definition für  $(x)_i$  angeben:

$$(x)_i = \mu_{z \leq x+1} (z, \neg [(p(i))^z \mid x] = 0) \dot{-} 1 \quad \text{falls } i > 0.$$

Wir müssen vom Ergebnis der beschränkten  $\mu$ -Rekursion 1 abziehen, weil wir den ersten Exponenten ermittelt haben, der *nicht mehr* Teiler ist.

Es fehlt noch der Fall  $i = 0$ : Da alle Potenzen von 1 immer 1 sind, macht es keinen Sinn vom Exponenten der 1 in der Primfaktorzerlegung zu sprechen. Daher setzen wir ihn einfach per Definition auf 0. Damit ergibt sich insgesamt

$$(x)_i = \begin{cases} \mu_{z \leq x+1} \left( z, \neg [(p(i))^z \mid x] = 0 \right) \dot{-} 1 & \text{falls } i > 0 \\ 0 & \text{falls } i = 0 \end{cases}$$

<sup>2</sup>Wir möchten den *beschränkten*  $\mu$ -Operator verwenden. Wie Sie wissen, ist der unbeschränkte  $\mu$ -Operator *nicht* primitiv-rekursiv.

Diese Funktion ist primitiv-rekursiv, da das „teilt“-Prädikat  $f|$ ,  $f >$ ,  $f =$ , sowie  $f \leq$ ,  $p$ , und die Potenzbildung in  $\mathcal{P}$  liegen und  $\mathcal{P}$  bezüglich  $\neg$ , Einsetzen, der beschränkten  $\mu$ -Rekursion und Fallunterscheidung abgeschlossen ist.<sup>3</sup>

## 7 Verschränkt rekursive Funktionen

Als Einleitung eine kleine ‚Mathematiker-Anekdote‘ zum Thema „Wie unendlich ist unendlich?“

*Stellen Sie sich vor, Sie leitet ein Hotel mit unendlich vielen Zimmern, bezeichnet mit den Zimmernummern 1, 2, 3 usw. Alle diese Zimmer seien schon belegt.<sup>a</sup>*

*Nun stellen Sie sich folgende Szenarien vor und versuchen Sie, eine Lösung zu finden:*

- (1) 1 zusätzlicher Gast  $a$  möchte einchecken.
- (2) Eine unendlich große Reisegruppe mit den durchnummerierten Anreisenden  $a_1, a_2, a_3, \dots$  möchte einchecken.
- (3) Unendlich viele unendlich große Reisegruppen mit Anreisenden

$a_{11}, a_{12}, a_{13}, \dots$

$a_{21}, a_{22}, a_{23}, \dots$

$\vdots$

*möchten einchecken.*

*Sie dürfen nun den [alten und neuen] Gästen sagen, welche Zimmer sie beziehen sollen. Da wir aber nur Einzelzimmer haben und unsere Gäste prude sind, braucht jeder sein eigenes Zimmer.*

—Gutenachtgeschichte für Mathematiker

„<sup>a</sup>Fragen sie uns nicht, wie sich unendlich viele Gäste am Frühstücksbuffet verhalten ...

Mögliche Lösungen folgen unten. Dabei seien die bisher eingetragenen Gäste  $G = \{g_1, g_2, g_3, \dots\}$  und  $Z : G \rightarrow \mathbb{N}$  die Abbildung, die jedem Gast seine Zimmernummer zuweist. Initial sei  $\forall i \ Z(g_i) = i$ .  $\curvearrowright$  jeder Gast hat genau ein Zimmer und alle Zimmer sind belegt ( $Z$  ist bijektiv)

<sup>3</sup>Mittlerweile sollte hier klar sein, wie man diese Konstruktion formal nach der Definition von  $\mathcal{P}$  führen kann. Wir empfehlen, sich das als Übung einmal aufzuschreiben.

- (1) Sage allen Gästen, sie sollen ein Zimmer weiterziehen, d. h. setze  $Z(g_i) := i + 1$  für alle  $i \in \mathbb{N}$ . Dann ist Zimmer 1 unbelegt! Also weisen wir es dem Neu-Ankömmling zu:  $Z(a) := 1$ . Das war einfach.
- (2) Um Ankömmlinge  $a_1, a_2, a_3, \dots$  unterzubringen, müssen wir *unendlich viele* Zimmer frei bekommen. Auch das geht einfach:

$$Z(x) := \begin{cases} 2i & \text{falls } x = g_i \\ 2i - 1 & \text{falls } x = a_i \end{cases}$$

d. h. wir schicken die ‚alten‘ Gäste in die geraden Zimmernummern und die ‚neuen‘ in die ungeraden.

- (3) Jetzt wird es schon kniffliger. Doch auch das ist kein Problem:

$$Z(x) := \begin{cases} 2^i & \text{falls } x = g_i \\ (p(i+1))^j & \text{falls } x = a_{ij} \end{cases}$$

Dabei sei  $p(i)$  die  $i$ -te Primzahl. Diese Abbildung ist injektiv, weil die Primfaktorzerlegung jeder Zahl  $Z(x)$  eindeutig ist. Man beachte: Wir haben jetzt nicht nur unendlich mal unendlich viele Gäste untergebracht; wir haben auch immer noch unendlich viele Zimmer frei [z. B. die Zimmer  $2 \cdot 3^i$  für alle  $i \geq 1$ ].

Fazit: Die natürlichen Zahlen bieten eine ganze Menge ‚Platz‘: Wenn man diesen geschickt nutzt, bekommt man unendlich oft die unendlich vielen natürlichen Zahlen in den natürlichen Zahlen selbst unter. Klar? Natürlich! ;-)

Weil Sie sich mittlerweile sicherlich fragt *Wozu das Ganze?*, fangen wir jetzt endlich mit der eigentlichen Aufgabe an.

Definiere

$$c : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\ c(x_1, x_2) = 2^{x_1} \cdot 3^{x_2}$$

Anschaulich codiert  $c$  ein Paar von natürlichen Zahlen in *einer* (!) natürlichen Zahl. Aufgrund der Eindeutigkeit der Primfaktorzerlegung (und da 2 und 3 Primzahlen sind) ist  $c$  injektiv. Das bedeutet aber, dass  $c$  *umkehrbar* ist.<sup>4</sup> Definiere dazu

$$d_1(y) := (y)_1 \\ d_2(y) := (y)_2$$

<sup>4</sup>Naja, streng genommen stimmt das nicht. Dazu müsste  $c$  schließlich bijektiv sein. Wenn wir aber den Definitionsbereich der gewünschten Umkehrfunktion auf  $\{2^i \cdot 3^j \mid i, j \geq 0\}$  beschränken, dann ist diese Funktion wohldefiniert. In der Anwendung, in der wir das  $c$  verwenden werden, ist diese Annahme immer erfüllt.

wobei  $(\cdot)_i$  die Funktion aus dem Buch (bzw. aus Abschnitt 6) ist, welche den Exponenten der  $i$ -te Primzahl liefert. Damit gilt offensichtlich

$$\forall (x_1, x_2) \quad \left( d_1(c(x_1, x_2)), d_2(c(x_1, x_2)) \right) = (x_1, x_2). \quad (2)$$

Außerdem sind  $c$ ,  $d_1$  und  $d_2$  direkt nach Konstruktion primitiv-rekursiv.

Nach dieser Vorarbeit komme ich jetzt auch wirklich zur eigentlichen Aufgabe, versprochen.

Behauptung: Die (verschränkt rekursiven) Funktionen  $f_1$  und  $f_2$  mit

$$\begin{aligned} f_1(0) &= 1 \\ f_2(0) &= 1 \\ f_1(x+1) &= f_1(x) + f_2(x) \\ f_2(x+1) &= f_1(x) \cdot f_2(x) \end{aligned}$$

sind primitiv-rekursiv.

Beweis: Definiere  $g$  mit

$$\begin{aligned} g(0) &= c(1, 1) \\ g(x+1) &= c\left(d_1(g(x)) + d_2(g(x)), d_1(g(x)) \cdot d_2(g(x))\right) \end{aligned}$$

und setze  $f'_1 = d_1 \circ g$  und  $f'_2 = d_2 \circ g$  [ $\circ$  sei Funktionskonkatenation].

$\left( \begin{array}{l} g \text{ berechnet die Werte für } f_1 \text{ und } f_2 \text{ synchron und codiert sie mittels } c. \text{ Wendet man also auf} \\ \text{das Ergebnis von } c \text{ die Dekodierfunktionen } d_1 \text{ bzw. } d_2 \text{ an, erhält man nach Gleichung (2)} \\ \text{die Einzelfunktionen } f_1 \text{ bzw. } f_2. \text{ Damit ist offensichtlich } f_1 = f'_1 \text{ und } f_2 = f'_2 \end{array} \right)$

$g$  ist durch primitive Rekursion aus Funktionen aus  $\mathcal{P}$  aufgebaut; damit ist  $g \in \mathcal{P}$  und schließlich auch  $f_1 = f'_1 \in \mathcal{P}$  und  $f_2 = f'_2 \in \mathcal{P}$ .  $\square$

Bemerkung: Die Fibonacci-Funktion lässt sich sehr ähnlich bauen  $\leadsto$  üben Sie es!





<http://www.springer.com/978-3-8348-1889-8>

Formale Grundlagen der Programmierung

Nebel, M.

2012, VIII, 194 S., Softcover

ISBN: 978-3-8348-1889-8