Kapitel 3

Rekursivität

3.1 Rekursive und GOTO-berechenbare Funktionen

Definition 3.1 *Die Menge der* rekursiven Funktionen *ist die kleinste Menge* $F \subseteq \bigcup_{k\in\mathbb{N}}\{f\colon \mathbb{N}^k \to \mathbb{N}\}$*, welche die folgenden Bedingungen erfüllt:*

- *F enthält die Nullfunktion* 0*, die Nachfolgerfunktion S und die Projektionsfunktion π_i^k ($1 \leq i \leq k$).*
- *F ist abgeschlossen unter Zusammensetzung. Das heißt, wenn die Funktionen $f\colon \mathbb{N}^\ell \to \mathbb{N}$ und $g_1, \ldots, g_\ell\colon \mathbb{N}^k \to \mathbb{N}$ alle in F liegen, dann liegt auch die durch*

  $$h(\bar{x}) = f(g_1(\bar{x}), \ldots, g_\ell(\bar{x}))$$

  gegebene Funktion $h\colon \mathbb{N}^k \to \mathbb{N}$ in F.
- *F ist abgeschlossen unter* primitiver Rekursion*. Das heißt, wenn die Funktionen $f\colon \mathbb{N}^k \to \mathbb{N}$ und $g\colon \mathbb{N}^{k+2} \to \mathbb{N}$ in F liegen, dann liegt auch die durch*

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

  gegebene Funktion $h\colon \mathbb{N}^{k+1} \to \mathbb{N}$ in F.
- *F ist abgeschlossen unter* μ-Rekursion*. Das heißt, wenn die Funktion $f\colon \mathbb{N}^{k+1} \to \mathbb{N}$ in F liegt und außerdem für alle $\bar{x} \in \mathbb{N}^k$ ein $y \in \mathbb{N}$ existiert, so dass $f(\bar{x}, y) > 0$ ist, dann liegt auch die durch*

  $$g(\bar{x}) = \mu y(f(\bar{x}, y) > 0) = \min\{y \in \mathbb{N} \mid f(\bar{x}, y) > 0\}$$

  gegebene Funktion $g\colon \mathbb{N}^k \to \mathbb{N}$ in F.

Bemerkung 3.2 *Alle primitiv rekursiven Funktionen sind rekursiv.*

Beweis Sei P die Menge aller primitiv rekursiven Funktionen und R die Menge aller rekursiven Funktionen. Sowohl P als auch R sind von der in Definition 1.1 betrachteten Art. Da P als die kleinste solche Menge definiert ist, gilt $P \subseteq R$. ∎

Definition 3.3 *Eine Menge $A \subseteq \mathbb{N}^k$ heißt* rekursiv*, wenn ihre charakteristische Funktion*

$$\begin{aligned} \chi_A\colon \mathbb{N}^k &\to \mathbb{N}, \\ \bar{x} &\mapsto \begin{cases} 1 & \text{falls } \bar{x} \in A \\ 0 & \text{sonst} \end{cases} \end{aligned}$$

rekursiv ist.

Bemerkung 3.4 *Eine Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ ist genau dann rekursiv, wenn ihr Graph $\Gamma_f \subseteq \mathbb{N}^{k+1}$ rekursiv ist.*

Beweis $\chi_=\colon \mathbb{N}^2 \to \mathbb{N}$, die charakteristische Funktion der Identität, ist sicher (primitiv) rekursiv. Wenn f rekursiv ist, dann ist wegen $\chi_{\Gamma_f}(\bar{x}, y) = \chi_=(f(\bar{x}), y)$ auch Γ_f rekursiv. Wenn umgekehrt Γ_f rekursiv ist, dann ist auch $f(\bar{x}) = \mu y(\chi_{\Gamma_f}(\bar{x}, y) > 0)$ rekursiv. ∎

Bemerkung 3.5 *Boolesche Kombinationen von rekursiven Mengen sind wieder rekursiv.*

Beweis Seien $A, B \subseteq \mathbb{N}^k$ rekursive Mengen. Dann gilt:

$$\begin{aligned}\chi_{A\cup B}(\bar{x}) &= \operatorname{sgn}(\chi_A(\bar{x}) + \chi_B(\bar{x}))\\ \chi_{A\cap B}(\bar{x}) &= \chi_A(\bar{x}) \cdot \chi_B(\bar{x})\\ \chi_{\mathbb{N}\setminus A}(\bar{x}) &= 1 \dot{-} \chi_A(\bar{x})\end{aligned}$$

Daher sind die charakteristischen Funktionen von $A \cup B$, $A \cap B$ und $\mathbb{N} \setminus A$ rekursiv. ∎

Bemerkung 3.6 *Durch Einsetzung von rekursiven Funktionen erhält man aus einer rekursiven Menge ebenfalls wieder eine rekursive Menge. Genauer: Wenn $A \subseteq \mathbb{N}^k$ eine rekursive Menge ist und $f_1, \ldots, f_k\colon \mathbb{N}^m \to \mathbb{N}$ rekursive Funktionen sind, dann ist auch $\{\bar{a} \in \mathbb{N}^m \mid (f_1(\bar{a}), \ldots, f_k(\bar{a})) \in A\}$ eine rekursive Menge.*

Beweis Die charakteristische Funktion dieser Menge ergibt sich durch Zusammensetzung von χ_A mit den Funktionen f_i. ∎

Bemerkung 3.7 *Wenn $f, g\colon \mathbb{N}^k \to \mathbb{N}$ rekursive Funktionen sind und $B \subseteq \mathbb{N}^k$ eine rekursive Menge ist, dann ist auch*

$$h(\bar{x}) = \begin{cases} f(\bar{x}) & \text{falls } \bar{x} \in B \\ g(\bar{x}) & \text{sonst} \end{cases}$$

eine rekursive Funktion.

Proposition 3.8 *Wenn $A \in \mathbb{N}^{k+1}$ und $f\colon \mathbb{N}^k \to \mathbb{N}$ rekursiv sind, dann sind auch folgende Mengen rekursiv:*

$$\{\bar{x} \in \mathbb{N}^k \mid \exists y < f(\bar{x})\big((\bar{x}, y) \in A\big)\}$$
$$\{\bar{x} \in \mathbb{N}^k \mid \forall y < f(\bar{x})\big((\bar{x}, y) \in A\big)\}$$

Jetzt können wir zeigen, dass es rekursive Funktionen gibt, die nicht primitiv rekursiv sind.

Satz 3.9 *Die dreistellige Funktion $\uparrow\colon \mathbb{N}^3 \to \mathbb{N}$, $(x, y, n) \mapsto x \uparrow^n y$ ist rekursiv.*

Beweis Die Funktion $\uparrow$ ist in einem intuitiven Sinn berechenbar. Die Beweisidee besteht darin, eine solche Berechnung, aus der sich $x \uparrow^n y = z$ ergibt, formal zu beschreiben. Die Berechnung wird aus Quadrupeln $(x, y, n, z) \in \Gamma_\uparrow$ bestehen, die wir durch die Zahlen $\langle x, y, n, z\rangle$ codieren. Die Quadrupel bilden insgesamt ein endliches Tupel, das wir wiederum mit der Funktion $\langle\cdot\rangle$ codieren. Für jedes Quadrupel der Berechnung fordern wir, dass entsprechend der rekursiven Definition von $\uparrow$ auch diejenigen Quadrupel in der Berechnung vorhanden sind, aus denen es folgt. Genauer:

$$\begin{aligned}S =& \{\langle x, y, n, z\rangle \mid n = 0 \text{ und } x \cdot y = z\}\\ &\cup \{\langle x, y, n, z\rangle \mid n > 0 \text{ und } y = 0 \text{ und } z = 1\} \subset \mathbb{N},\end{aligned}$$

$$R = \{(\langle x, y+1, n+1, z\rangle, \langle x, y, n+1, u\rangle, \langle x, u, n, z\rangle) \mid x, y, n, z, u \in \mathbb{N}\} \subset \mathbb{N}^3.$$

$$\begin{aligned}B = \{b \in \mathbb{N} \mid \forall i < \operatorname{Lg}(b)\big(&\\ &\operatorname{Component}(b, i) \in S \text{ oder } \exists s < \operatorname{Lg}(b) \exists t < \operatorname{Lg}(b)\big(\\ &(\operatorname{Component}(b, i), \operatorname{Component}(b, s), \operatorname{Component}(b, t)) \in R\\ &\big)\big)\} \subseteq \mathbb{N}.\end{aligned}$$

Man überlegt sich leicht, dass S, R und B primitiv rekursiv sind, und dass B die Menge der Berechnungen b im obigen Sinne ist. Daher können wir die Funktion $\uparrow$ wie folgt beschreiben:

$$\beta(x,y,n) = \mu b\Big(b \in B \text{ und } \exists i < \mathrm{Lg}(b) \exists z < b\big(\,\mathrm{Component}(b,i) = \langle x,y,n,z\rangle\,\big)\Big)$$
$$\gamma(x,y,n,b) = \mu i\Big(\exists z\big(\,\mathrm{Component}(b,i) = \langle x,y,n,z\rangle\,\big)\Big)$$
$$x \uparrow^n y = \mathrm{Component}\,\big(\,\mathrm{Component}(\beta(x,y,n), \gamma(x,y,n,\beta(x,y,n))), 3\big)$$

$\beta(x,y,n)$ gibt uns den numerisch kleinsten Code einer Berechnung, in der das Quadrupel $\langle x,y,n,x\uparrow^n y\rangle$ vorkommt.[1] Dann müssen wir nur noch in der Berechnung den Wert von $x \uparrow^n y$ nachschlagen. Man überlegt sich jetzt leicht, dass β und folglich auch $\uparrow$ rekursiv ist. ∎

Für eine alternative Beschreibung der rekursiven Funktionen benutzen wir Programme in einer Programmiersprache, die Ähnlichkeiten mit der früher sehr verbreiteten Sprache BASIC hat. GOTO-Programme sind definiert wie LOOP-Programme, nur dass wir an Stelle von Schleifen bedingte Sprunganweisungen erlauben. Eine Neuerung hierbei ist, dass ein GOTO-Programm aus mit 0 beginnend fortlaufend durchnummerierten Zeilen besteht. In jeder Zeile kann wie üblich einer der folgenden vertrauten Befehle stehen.

- `y = Zero()`
- `y = Val(x)`
- `y = Inc(x)`
- `y = Dec(x)`

Allerdings gibt es in GOTO-Programmen keine Schleifen im bisherigen Sinne. Diese sind durch eine flexiblere Anweisung der Art `if x goto 42` ersetzt. Bei der Ausführung des Programms bewirkt diese Anweisung einen Sprung in die Zeile 42, falls der Inhalt des durch `x` bezeichneten Registers nicht die Zahl 0 ist. Dabei kann natürlich an Stelle von `x` wieder ein beliebiger Variablenname stehen und an Stelle von 42 eine beliebige natürliche Zahl. Ein GOTO-Programm hält an, sobald es eine Zeile ohne Anweisung erreicht.

Die GOTO-Programme erlauben es uns, sogenannten *Spagetti-Code* zu schreiben, also im schlimmsten Fall Programme, die ohne erkennbare Struktur kreuz und quer springen. Man kann aber zeigen (siehe Übungsaufgaben), dass die einzige wirklich wesentliche Erweiterung gegenüber LOOP-Programmen darin besteht, dass wir nun Schleifen implementieren können, bei denen nicht mehr am Anfang feststeht, wie oft sie durchlaufen werden. Insbesondere können wir jetzt auch Schleifen implementieren, die gar nicht enden:

```
0   false = Zero()
1   true = Inc(false)
2   output = Zero()
3   output = Inc(output)
4   if true goto 3
```

Dieses Programm läuft unendlich lange, und bei jedem Durchgang durch den Schleifenkörper ändert sich der Wert von `output`. Daher hat es keinen Sinn, diesem Programm eine Funktion zuzuschreiben, die dadurch berechnet wird.

Wir „lösen“ dieses Problem, indem wir von *partiellen (k-stelligen) Funktionen* sprechen, das heißt, Funktionen, deren Definitionsbereich nur eine Teilmenge von $\mathbb{N}^k$ ist.

Definition 3.10 *Eine* partielle Funktion $f\colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ *ist eine Funktion* $f\colon X \to \mathbb{N}$, *wobei* $X \subseteq \mathbb{N}^k$.

Wenn einfach nur von einer „Funktion“ die Rede ist, ist aber nach wie vor immer eine totale Funktion gemeint.

Definition 3.11 *Sei* $k \in \mathbb{N}$. *Eine partielle Funktion* $f\colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ *heißt* GOTO-berechenbar, *falls es es ein GOTO-Programm gibt, welches die Funktion im folgenden Sinne berechnet. Falls das Programm zu Beginn in den Registern* `input_1`, `input_2`, …, `input_`k *die Zahlen* $x_1, x_2, \ldots, x_k \in \mathbb{N}$ *stehen hat und in allen anderen Registern die Zahl 0, dann passiert Folgendes:*

[1] Es ist nicht wichtig, dass es der kleinste Code ist. Es kommt nur darauf an, dass es immer eine solche Berechnung gibt und wir von β den Code einer solchen erhalten.

- *Falls $f(x_1, x_2, \ldots, x_k)$ definiert ist, dann hält das Programm an, und am Ende steht im Register* `output` *die Zahl $f(x_1, x_2, \ldots, x_k)$.*
- *Falls $f(x_1, x_2, \ldots, x_k)$ nicht definiert ist, dann hält das Programm nicht an.*

Zwei GOTO-Programme heißen äquivalent, wenn sie für jedes $k \in \mathbb{N}$ dieselbe partielle k-stellige Funktion $f\colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ berechnen.

Das folgende GOTO-Programm berechnet beispielsweise das Produkt von zwei natürlichen Zahlen.

```
0   false = Zero()
1   true = Inc(false)
2   i = Val(input_1)
3   output = Zero()
4   if i goto 6
5   if true goto 14
6   j = Val(input_2)
7   if j goto 9
8   if true goto 12
9   output = Inc(output)
10  j = Dec(j)
11  if true goto 7
12  i = Dec(i)
13  if true goto 4
```

Das Programm ist leichter zu verstehen, wenn man sich klarmacht, dass es sich einfach um eine Übersetzung des folgenden LOOP-Programms handelt.

```
output = Zero()
do input_1 times {
  do input_2 times {
    output = Inc(output)
  }
}
```

GOTO-Orakelprogramme sind auf die offensichtliche Weise definiert.

Lemma 3.12 *Jede Funktion, die durch ein GOTO-Orakelprogramm berechenbar ist, wobei jeder Orakelname durch eine GOTO-berechenbare Funktion interpretiert wird, ist selbst GOTO-berechenbar.*

Beweis Man kann sich leicht überlegen, dass man jede Zeile, die ein Orakel benutzt, durch das (leicht abgewandelte) GOTO-Programm ersetzen kann, das diese interpretiert. Die hinteren Zeilen ändern dabei ihre Zeilennummern, so dass die GOTO-Sprünge ebenfalls angepasst werden müssen. ∎

Lemma 3.13 *Jede rekursive Funktion ist GOTO-berechenbar.*

Beweis Der Beweis geht im Grunde genauso wie der Beweis von Satz 1.5. Wir müssen zeigen, dass die GOTO-berechenbaren (totalen) Funktionen eine Menge F wie in der Definition der rekursiven Funktionen bilden. Wir führen hier nur den Abschluss unter μ-Rekursion aus.

Sei also eine GOTO-berechenbare Funktion $f\colon \mathbb{N}^{k+1} \to \mathbb{N}$ gegeben, so dass $g(\bar{x}) = \mu y(f(\bar{x}, y) > 0)$ ebenfalls eine totale Funktion ist.

```
0   y = Zero()
1   z = F(input_1, ..., input_k, y)
2   y = Inc(y)
3   if z goto 5
4   if y goto 1
5   output = Dec(y)
```

Man macht sich leicht klar: Wenn man den $(k+1)$-stelligen Orakelnamen F im obigen GOTO-Orakelprogramm durch f interpretiert, dann berechnet das Programm g. ∎

Übungsaufgaben

Übungsaufgabe 3.1 *Eine Menge $A \subseteq \mathbb{N}^k$ heißt primitiv rekursiv, wenn ihre charakteristische Funktion primitiv rekursiv ist. Sei $f\colon \mathbb{N} \to \mathbb{N}$ eine Funktion, deren Graph Γ_f primitiv rekursiv ist. Zeigen Sie, dass f rekursiv ist, aber im Allgemeinen nicht primitiv rekursiv.*

Übungsaufgabe 3.2 *Schreiben Sie ein GOTO-Programm, das die dreistellige Hyperoperatorfunktion berechnet. Können Sie das auch mit einem LOOP-Programm? Warum ist das so viel schwerer?*

Übungsaufgabe 3.3 *WHILE-Programme sind definiert wie LOOP-Programme, nur dass zusätzlich noch die folgende neue Art von Schleifen erlaubt ist.*

```
while x do {
  ...
}
```

Wieder kann `x` *durch einen beliebigen Variablennamen ersetzt werden, und die drei Punkte stehen für ein beliebiges WHILE-Programm. Der Schleifenkörper wird nur dann ausgeführt, wenn der Wert der Variable* `x` *nicht Null ist. Nach jeder Ausführung des Schleifenkörpers wird* `x` *aber erneut inspiziert. Wenn der Wert dann immer noch nicht Null ist, wird die Schleife erneut ausgeführt.*

WHILE-Programme sind näher an modernen Programmiersprachen als GOTO-Programme. Sie sind in der Regel übersichtlicher und leichter zu verstehen.

Zeigen Sie: Zu jedem WHILE-Programm gibt es ein äquivalentes WHILE-Programm, in welchem nur die neue Art von Schleifen vorkommt. Zu jedem WHILE-Programm gibt es ein äquivalentes GOTO-Programm.

Übungsaufgabe 3.4 *Betrachten Sie Kontrollstrukturen der folgenden Art, mit der offensichtlichen Bedeutung:*

```
if x then {
  ...
} else {
  ...
}
```

Wie kann man WHILE-Programme in der so erweiterten Sprache in GOTO-Programme übersetzen? Wie kann man diese Kontrollstruktur mit Hilfe von `loop x times` *ausdrücken?*

Übungsaufgabe 3.5 *Zu jedem GOTO-Programm gibt es ein äquivalentes WHILE-Programm. (Hinweis: Es geht mit nur einer einzigen WHILE-Schleife! Benutzen Sie Orakel und Aufgabe 3.4, und führen Sie in einer Variable Buch über die Zeile des GOTO-Programms, in der sich Ihr WHILE-Programm gerade befindet.)*

Übungsaufgabe 3.6 *Den Begriff der Rekursivität kann man auch auf partielle Funktionen ausweiten, indem man bei der μ-Rekursion auf die Bedingung verzichtet, die die Totalität sicherstellt. Welche weiteren Anpassungen muss man dann in Definition 3.1 noch machen? Zeigen Sie: Jede rekursive Funktion ist eine rekursive partielle Funktion, und jede rekursive partielle Funktion ist GOTO-berechenbar. Mit welchem Resultat aus dem nächsten Abschnitt kann man daraus folgern, dass jede rekursive partielle Funktion, die außerdem eine (totale) Funktion ist, eine rekursive Funktion ist?*

3.2 Äquivalenz von Rekursivität und GOTO-Berechenbarkeit

Ab jetzt werden wir stillschweigend voraussetzen, dass in jedem GOTO-Programm außer der Variable `output` nur Variablen der Form `input_1`, `input_2`, `input_3`, ..., `input_9`, `input_10`, `input_11`, ... vorkommen. Dadurch können wir in offensichtlicher Weise jeden Variablennamen s durch eine natürliche Zahl $\ulcorner s \urcorner$ codieren: $\ulcorner$`output`$\urcorner = 0$, $\ulcorner$`input_1`$\urcorner = 1$, $\ulcorner$`input_2`$\urcorner = 2$ usw.

Bemerkung 3.14 *Jedes GOTO-Programm kann in ein äquivalentes GOTO-Programm transformiert werden, das diese Bedingung erfüllt.*

Eine Zeile eines GOTO-Programms codieren wir wie folgt als natürliche Zahl:

- $\ulcorner$`y = Zero()`$\urcorner = \langle 1, 0, \ulcorner$`y`$\urcorner \rangle$
- $\ulcorner$`y = Val(x)`$\urcorner = \langle 2, \ulcorner$`x`$\urcorner, \ulcorner$`y`$\urcorner \rangle$
- $\ulcorner$`y = Inc(x)`$\urcorner = \langle 3, \ulcorner$`x`$\urcorner, \ulcorner$`y`$\urcorner \rangle$
- $\ulcorner$`y = Dec(x)`$\urcorner = \langle 4, \ulcorner$`x`$\urcorner, \ulcorner$`y`$\urcorner \rangle$
- $\ulcorner$**`if x goto`** $\ell \urcorner = \langle 5, \ulcorner$`x`$\urcorner, \ell \rangle$

Hierbei stehen `x`, `y` für Variablennamen, die den eben beschriebenen Einschränkungen entsprechen (so dass $\ulcorner$`x`$\urcorner$ und $\ulcorner$`y`$\urcorner$ auch definiert sind), und ℓ steht für eine natürliche Zahl bzw. im Programmcode für ihre Dezimaldarstellung.

Durch diese Codierung sind wir richtigen Mikroprozessoren etwas nähergekommen. Wir können uns einen idealisierten Mikroprozessor vorstellen, mit fortlaufend durchnumerierten Registern, von denen jedes eine natürliche Zahl fassen kann. Wir codieren ein Programm mit L Zeilen durch die Zahl $\mathcal{P} = \langle p_0, p_1, \ldots, p_{L-1} \rangle$, wobei $p_i \in \mathbb{N}$ der Code für die i-te Zeile ist.

Wir schauen uns nun die Ausführung eines Programms im Detail an. Dabei interessieren wir uns für den Zeitpunkt, wenn der Mikroprozessor gerade die Zeile gewechselt, die neue Zeile aber noch nicht ausgeführt hat. Zu jedem solchen Zeitpunkt enthält jedes Register i eine natürliche Zahl $x_i \in \mathbb{N}$ und der Programmzähler befindet sich in der als nächstes auszuführenden Zeile $\ell \in \mathbb{N}$. Wir codieren den Zustand der Register durch die Zahl $\mathcal{R} = \langle x_0, x_1, \ldots, x_{R-1} \rangle$ und den gesamten Zustand des Mikroprozessors durch $\mathcal{Z} = \langle \ell, \mathcal{R} \rangle$.

Wir benutzen von jetzt an die Schreibweise $(x)_i = \mathrm{Component}(x, i)$. Der jeweils nächste Zustand ist dann durch die folgende Funktion gegeben:

$$N_{\mathcal{P}}(\mathcal{Z}) = \begin{cases} \langle \ell + 1, \mathrm{Replace}(\mathcal{R}, y, 0) \rangle & \text{falls } c = 1 \\ \langle \ell + 1, \mathrm{Replace}(\mathcal{R}, y, (\mathcal{R})_x) \rangle & \text{falls } c = 2 \\ \langle \ell + 1, \mathrm{Replace}(\mathcal{R}, y, (\mathcal{R})_x + 1) \rangle & \text{falls } c = 3 \\ \langle \ell + 1, \mathrm{Replace}(\mathcal{R}, y, (\mathcal{R})_x \dot{-} 1) \rangle & \text{falls } c = 4 \\ \langle \ell + 1, \mathcal{R} \rangle & \text{falls } c = 5 \text{ und } (\mathcal{R})_x = 0 \\ \langle y, \mathcal{R} \rangle & \text{falls } c = 5 \text{ und } (\mathcal{R})_x \neq 0 \\ \mathcal{Z} & \text{sonst,} \end{cases}$$

wobei wir zur besseren Lesbarkeit die folgenden Abkürzungen verwendet haben:

$$\ell = (\mathcal{Z})_0 \qquad \mathcal{R} = (\mathcal{Z})_1 \qquad C = (\mathcal{P})_\ell$$
$$c = (C)_0 \qquad x = (C)_1 \qquad y = (C)_2$$

Diese Funktion $N \colon \mathbb{N}^2 \to \mathbb{N}, (\mathcal{P}, \mathcal{Z}) \mapsto N_{\mathcal{P}}(\mathcal{Z})$ ist primitiv rekursiv. Natürlich ist auch für jedes $k \in \mathbb{N}$ die durch

$$A^k(x_1, \ldots, x_k) = \langle 0, \langle 0, x_1, \ldots, x_k \rangle \rangle$$

gegebene Funktion $A^k \colon \mathbb{N}^k \to \mathbb{N}$ primitiv rekursiv. Diese Funktion gibt uns den anfänglichen Zustand bei einer Berechnung, in Abhängigkeit von den Eingabewerten.

Wir halten fest, was wir bisher herausgefunden haben.

Lemma 3.15 *Es gibt primitiv rekursive Funktionen $N \colon \mathbb{N}^2 \to \mathbb{N}$ und $A^k \colon \mathbb{N}^k \to \mathbb{N}$ sowie für jedes GOTO-Programm eine Zahl $\mathcal{P} \in \mathbb{N}$, so dass Folgendes gilt.*

Sei $f \colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ die durch das Programm berechnete partielle Funktion und seien $x_1, \ldots, x_k \in \mathbb{N}$. Wenn s die kleinste Zahl ist, so dass für $\mathcal{Z} = N_{\mathcal{P}}^s(A^k(x_1, \ldots, x_k))$ und $\ell = (\mathcal{Z})_0$ die Bedingung $(\mathcal{P})_\ell = 0$ erfüllt ist, dann ist $f(x_1, \ldots, x_k) = (\mathcal{R})_0$, wobei $\mathcal{R} = (\mathcal{Z})_1$. Wenn es keine solche Zahl s gibt, dann ist auch $f(x_1, \ldots, x_k)$ nicht definiert.

Kleenesche Normalform

Satz 3.16 *Es gibt eine primitiv rekursive Funktion $\mathcal{U}\colon \mathbb{N} \to \mathbb{N}$ und für jedes $k \in \mathbb{N}$ eine primitiv rekursive Funktion $\mathcal{T}_k\colon \mathbb{N}^{k+2} \to \mathbb{N}$, so dass Folgendes gilt:*

Für jede rekursive Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ existiert eine Zahl $e \in \mathbb{N}$, so dass für alle $x_1, \ldots, x_k \in \mathbb{N}$ ein $y \in \mathbb{N}$ existiert, so dass $\mathcal{T}_k(e, x_1, \ldots, x_k, y) > 0$ ist, und es gilt

$$f(x_1, \ldots, x_k) = \mathcal{U}\Big(\mu y\big(\mathcal{T}_k(e, x_1, \ldots, x_k, y) > 0\big)\Big).$$

Da alle rekursiven Funktionen GOTO-berechenbar sind, folgt der Satz sofort aus dem folgenden Lemma.

Lemma 3.17 *Es gibt eine primitiv rekursive Funktion $\mathcal{U}\colon \mathbb{N} \to \mathbb{N}$ und für jedes $k \in \mathbb{N}$ eine primitiv rekursive Funktion $\mathcal{T}_k\colon \mathbb{N}^{k+2} \to \mathbb{N}$, so dass Folgendes gilt:*

Wenn $\mathcal{P}$ ein GOTO-Programm codiert, dann hat die durch das Programm definierte partielle k-stellige Funktion $f\colon \mathbb{N}^k \dashrightarrow \mathbb{N}$ die Form

$$f(x_1, \ldots, x_k) = \mathcal{U}\Big(\mu y\big(\mathcal{T}_k(\mathcal{P}, x_1, \ldots, x_k, y) > 0\big)\Big).$$

(Insbesondere existiert $\mu y(\mathcal{T}_k(\mathcal{P}, x_1, \ldots, x_k, y) > 0)$ genau dann, wenn $f(x_1, \ldots, x_k)$ definiert ist.)

Beweis Mit Hilfe der Funktionen aus Lemma 3.15 definieren wir

$$\mathcal{T}_k^*(\mathcal{P}, x_1, \ldots, x_k, y, 0) = \begin{cases} 1 & \text{falls } (y)_0 = A^k(x_1, \ldots, x_k) \\ 0 & \text{sonst,} \end{cases}$$

$$\mathcal{T}_k^*(\mathcal{P}, x_1, \ldots, x_k, y, n+1) = \begin{cases} 1 & \text{falls } \mathcal{T}_k^*(\mathcal{P}, x_1, \ldots, x_k, y, n) > 0 \\ & \text{und } (y)_{n+1} = N_{\mathcal{P}}((y)_n) \\ 0 & \text{sonst.} \end{cases}$$

Man sieht leicht, dass die so definierte Funktion primitiv rekursiv ist. Außerdem macht man sich leicht klar, dass genau dann $\mathcal{T}_k^*(\mathcal{P}, x_1, \ldots, x_k, y, \lg y) > 0$ gilt, wenn y eine anfängliche Folge von Zuständen codiert, die bei der Ausführung des Programms mit Startwerten $(x_1, \ldots, x_k)$ in auftreten. Wir definieren nun $\mathcal{T}_k$ selbst wie folgt.

$$\mathcal{T}_k(\mathcal{P}, x_1, \ldots, x_k, y) = \begin{cases} 1 & \text{falls } \mathcal{T}_k^*(\mathcal{P}, x_1, \ldots, x_k, y, \lg y) > 0 \text{ und } (\mathcal{P})_\ell = 0, \\ & \text{wobei } \ell = (\mathcal{Z})_0 \text{ und } \mathcal{Z} = (y)_{\lg y \dot{-} 1} \\ 0 & \text{sonst,} \end{cases}$$

Wenn das Programm bei den Eingaben $(x_1, \ldots, x_k)$ anhält, dann gibt es ein y, so dass $\mathcal{T}_k(\mathcal{P}, x_1, \ldots, x_k, y) > 0$ ist. (Und wenn es das Programm nicht anhält, gibt es gar kein solches y.) Für jedes solche y ist $\mathcal{Z} = (y)_{\lg y \dot{-} 1}$ der Code des letzten Zustands, $\mathcal{R} = (\mathcal{Z})_1$ codiert die Registerinhalte des letzten Zustands, und $(\mathcal{R})_0 = ((\mathcal{Z})_1)_0 = \Big(\Big((y)_{\lg y \dot{-} 1}\Big)_1\Big)_0$ ist der Inhalt des Ausgaberegisters am Ende. Also genügt es,

$$\mathcal{U}(y) = \Big(\Big((y)_{\lg y \dot{-} 1}\Big)_1\Big)_0$$

zu wählen. ∎

Korollar 3.18 *Die rekursiven Funktionen sind genau die GOTO-berechenbaren (totalen) Funktionen.*

Darüberhinaus zeigt die Kleenesche Normalform, dass man jede rekursive Funktion durch Komposition, primitive Rekursion und eine einzige Anwendung der μ-Rekursion aus den Grundfunktionen erhalten kann.

Übungsaufgaben

Übungsaufgabe 3.7 *Für jedes $k \in \mathbb{N}$ gibt es ein WHILE-Programm U mit der folgenden Eigenschaft. Für jedes GOTO-Programm P existiert eine natürliche Zahl $p \in \mathbb{N}$, so dass gilt: Wenn $f_U\colon \mathbb{N}^{k+1} \dashrightarrow \mathbb{N}$ die durch U berechnete partielle Funktion ist, dann ist die durch P berechnete partielle Funktion f_P gegeben durch $f_P(x_1, \ldots, x_k) = f_U(x_1, \ldots, x_k, p)$. (Hinweis: Wie Aufgabe 3.5. Benutzen Sie die beschriebene Codierung, um das ganze Programm in einer einzigen Variable halten zu können.)*

Übungsaufgabe 3.8 *Zeigen Sie mit Hilfe des Kleene-Prädikats direkt (also ohne Aufgabe 3.5 oder 3.7 zu benutzen): Jede GOTO-berechenbare partielle Funktion ist WHILE-berechenbar.*

3.3 Rekursive Aufzählbarkeit und das Halteproblem

Definition 3.19 *Eine Menge $A \subseteq \mathbb{N}^k$ heißt* rekursiv aufzählbar, *falls A Projektion einer rekursiven Menge $\bar{A} \subseteq \mathbb{N}^{k+1}$ ist: $(x_1, \ldots, x_k) \in A$ genau dann, wenn es ein y gibt, so dass $(x_1, \ldots, x_k, y) \in \bar{A}$.*

Bemerkung 3.20 *Alle rekursiven Mengen sind rekursiv aufzählbar. Die rekursiv aufzählbaren Mengen sind abgeschlossen unter Projektion.*

Beweis Dass jede rekursive Menge rekursiv aufzählbar ist, sieht man einfach mit Hilfe von Projektionen und Zusammensetzung. Jede rekursiv aufzählbare Menge $A \subseteq \mathbb{N}^{k+1}$ hat laut Definition die Form $A = \{(\bar{x}, y) \mid \exists z : (\bar{x}, y, z) \in \bar{A}\}$, wobei $\bar{A} \subseteq \mathbb{N}^{k+1}$ rekursiv ist. Die Projektion von A hat daher die Form

$$\{\bar{x} \in \mathbb{N}^k \mid \exists y : (\bar{x}, y) \in A\} = \{\bar{x} \mid \exists y \exists z : (\bar{x}, y, z) \in \bar{A}\} = \{\bar{x} \mid \exists u : (\bar{x}, L(u), R(u)) \in \bar{A}\},$$

wobei L und R die beiden Inversen der Cantorschen Paarungsfunktion J sind. ∎

Proposition 3.21 *Die folgenden Bedingungen an eine Menge $A \subseteq \mathbb{N}^k$ sind äquivalent.*

1. *A ist Projektion einer Menge, deren charakteristische Funktion primitiv rekursiv ist (einer primitiv rekursiven Menge).*
2. *A ist rekursiv aufzählbar.*
3. *A ist Projektion einer rekursiv aufzählbaren Menge.*

Beweis $1 \Rightarrow 2 \Rightarrow 3$ ist trivial. $3 \Rightarrow 2$ heißt einfach, dass die Projektion einer rekursiv aufzählbaren Menge wieder rekursiv aufzählbar ist.

$2 \Rightarrow 1$: Sei $A \subseteq \mathbb{N}^k$ eine rekursiv aufzählbare Menge, also $A = \{\bar{x} \mid \exists y : (\bar{x}, y) \in \bar{A}\}$ für eine rekursive Menge $\bar{A} \subseteq \mathbb{N}^{k+1}$. Nach Satz 3.16 gilt $\chi_{\bar{A}}(\bar{x}) = \mathcal{U}(\mu u(\mathcal{T}_{k+1}(e, \bar{x}, y, u) > 0))$, wobei $\mathcal{T}_{k+1}$ und $\mathcal{U}$ primitiv rekursive Funktionen sind und e eine Konstante ist. Es folgt $A = \{\bar{x} \mid \exists y : (\bar{x}, y) \in \bar{A}'\}$, wobei $\bar{A}' \subseteq \mathbb{N}^{k+1}$ definiert ist durch

$$\chi_{\bar{A}'}(\bar{x}, y) = \begin{cases} 1 & \text{falls } \mathcal{T}_{k+1}(e, \bar{x}, y, u) > 0 \text{ und } (\forall z < y : \mathcal{T}_{k+1}(e, \bar{x}, z, u) = 0) \text{ und } \mathcal{U}(y) = 1, \\ 0 & \text{sonst.} \end{cases}$$

Man sieht mit den Ergebnissen von Kapitel 1 leicht, dass $\chi_{\bar{A}'}$ primitiv rekursiv ist. ∎

Proposition 3.22 *Die folgenden Bedingungen an eine Menge $A \subseteq \mathbb{N}^k$ sind äquivalent.*

1. *A ist rekursiv aufzählbar.*
2. *$A = \emptyset$ oder A ist das Bild einer kompenentenweise primitiv rekursiven Funktion $f \colon \mathbb{N} \to \mathbb{N}^k$.*
3. *$A = \emptyset$ oder A ist das Bild einer komponentenweise rekursiven Funktion $f \colon \mathbb{N} \to \mathbb{N}^k$.*

Beweis $1 \Rightarrow 2$: Wenn $A \subseteq \mathbb{N}$ rekursiv aufzählbar ist, dann ist A von der Form $A = \{x \mid \exists y : (x, y) \in \bar{A}\}$ für eine primitiv rekursive Menge $\bar{A}$. Sei $a_0 \in A$ beliebig. Dann ist A das Bild der primitiv rekursiven Funktion

$$f(x, y) = \begin{cases} x & \text{falls } (x, y) \in \bar{A}, \\ a_0 & \text{sonst.} \end{cases}$$

Das klappt nur dann nicht, wenn es wegen $A = \emptyset$ kein solches a_0 gibt. Auf dieselbe Weise erhalten wir für $k > 1$ eine komponentenweise primitiv rekursive Funktion $f \colon \mathbb{N}^k \to \mathbb{N}^k$, die sich mit Hilfe der Umkehrungen der Cantorschen Paarungsfunktion zu einer Funktion $f' \colon \mathbb{N} \to \mathbb{N}^k$ umschreiben lässt. $2 \Rightarrow 3$ ist trivial.

$3 \Rightarrow 1$: Klar für $A = \emptyset$. Wenn $A \subseteq \mathbb{N}$ das Bild einer rekursiven Funktion f ist, dann gilt $A = \{x \in \mathbb{N} \mid \exists u : f(u) = x\}$. Durch die Bedingung $f(u) = x$ wird eine rekursive Menge definiert (vgl. Bemerkung3.4). ∎

Bemerkung 3.23 *Eine Menge ist genau dann rekursiv, wenn sie selbst und ihr Komplement rekursiv aufzählbar sind.*

Beweis Sei $A \subseteq \mathbb{N}^k$ rekursiv aufzählbar, also $A = \{\bar{x} \mid \exists y : (\bar{x}, y) \in \bar{A}\}$ für eine rekursive Menge $\bar{A} \subseteq \mathbb{N}^{k+1}$. Sei außerdem auch $B = \mathbb{N}^k \setminus A$ rekursiv aufzählbar, also $B = \{\bar{x} \mid \exists y : (\bar{x}, y) \in \bar{B}\}$ für eine rekursive Menge $\bar{B} \subseteq \mathbb{N}^{k+1}$. Dann können wir eine rekursive Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ definieren durch $f(\bar{x}) = \mu y((\bar{x}, y) \in \bar{A}$ oder $(\bar{x}, y) \in \bar{B})$.[2] Damit ist $\chi_A(\bar{x}) = \chi_{\bar{A}}(\bar{x}, f(\bar{x}))$ eine rekursive Funktion. ∎

Satz 3.24 *Es gibt eine universelle rekursiv aufzählbare Menge $W \subseteq \mathbb{N}^2$: W selbst ist rekursiv aufzählbar, und für jede rekursiv aufzählbare Menge $A \subseteq \mathbb{N}$ gibt es ein $c \in \mathbb{N}$, so dass $A = \{x \in \mathbb{N} \mid (c, x) \in W\}$.*

Beweis Die Menge

$$W = \{(c, x) \in \mathbb{N}^2 \mid \exists y(\mathcal{T}_1(c, x, y) > 0)\}$$

tut es. Weil $\mathcal{T}_1$ (primitiv) rekursiv ist, ist W rekursiv aufzählbar. Um die Universalität zu zeigen, betrachten wir eine beliebige rekursiv aufzählbare Menge $A = \{x \in \mathbb{N} \mid \exists z((x, z) \in \bar{A}\}$, wobei $\bar{A}$ rekursiv ist. Sei $\mathcal{P}$ der Code eines GOTO-Programms, das $\mu z(\chi_{\bar{A}}(x, z) > 0)$ berechnet. Das Programm hält genau dann an, wenn $x \in A$ ist. Daher ist $A = \{x \in \mathbb{N} \mid (\mathcal{P}, x) \in W\}$. ∎

Korollar 3.25 *W ist rekursiv aufzählbar, aber nicht rekursiv.*

Beweis Wenn W rekursiv wäre, dann wäre auch $A = \{x \in \mathbb{N} \mid (x, x) \notin W\}$ rekursiv und insbesondere rekursiv aufzählbar. Sei $c \in \mathbb{N}$ so, dass $A = \{x \in \mathbb{N} \mid (c, x) \in W\}$. Es folgt $c \in A \iff c \notin A$, also ein Widerspruch. ∎

Übungsaufgaben

Übungsaufgabe 3.9 *Eine Menge $A \subseteq \mathbb{N}^k$ ist genau dann rekursiv aufzählbar, wenn sie der Definitionsbereich einer GOTO-berechenbaren partiellen Funktion ist.*

[2] Genaugenommen: $f(\bar{x}) = \mu y\big(\chi_{\bar{A}}(\bar{x}, y) + \chi_{\bar{B}}(\bar{x}, y) > 0\big)$.