

Primitive Rekursion

Alexander Hölzle

14.01.2007

Inhaltsverzeichnis

Motivation	i
1 Rekursive Funktionen	1
1.1 Nicht berechenbare Funktionen	1
1.2 Primitiv rekursive Funktionen	2
1.3 Arithmetische Funktionen	4
1.4 Primitiv rekursive Relationen	8
1.5 μ -Operator	11
1.5.1 Unbeschränkte μ -Operator	12
1.5.2 Beschränkte μ -Operator	14
2 Die Ackermann-Funktion	16

Motivation, Überblick und Notation

Der Begriff der *Berechenbarkeit* spielt neben dem der Komplexität in der klassischen theoretischen Informatik eine zentrale Rolle. Zunächst sollte man sich fragen, was das Adjektiv „*berechenbar*“ überhaupt bedeutet und worauf sich dieser Begriff bezieht. Intuitiv werden viele darunter wohl verstehen, dass etwas „sich errechnen lässt“ bzw. „vorhersehbar“ ist. Um jedoch etwas berechnen zu können benötigt man sicherlich ein Kochrezept, also einen Algorithmus bzw. eine Funktionsvorschrift. Schließlich müssen wir auch noch klären, *was* man alles berechnen kann, es wird sich nämlich zeigen, dass nicht alle Funktionen berechenbar sind.

Sowohl Turing- als auch Registermaschinen sind Modelle von Maschinen. Sie nähern sich der Frage nach der Berechenbarkeit von der Seite der berechenbaren Automaten. Die jeweiligen Programme, welche auf diesen Maschinen laufen, müssen spezielle für das jeweilige Modell und die jeweilige Problemstellung zugeschnitten sein.

Einen hiervon differierten (aber äquivalenten) Zugang zur Frage der Berechenbarkeit basiert auf speziellen Funktionenklassen und darauf abgeschlossenen Operationen (Verknüpfungen). Mit den so genannte rekursiven Funktionen werden wir *Lösungen von Problemen* beschreiben und nicht, wie es bei den Maschinenmodellen üblich ist, die einzelnen Schritte des Rechenweges. Das Modell der rekursiven Funktionen versucht also *nicht* eine Maschine bzw. einen Automaten nachzubilden, sondern packt das Problem der Berechenbarkeit sozusagen bei der Wurzel, d.h. es nähert sich der Frage nach der Berechenbarkeit von Seite der Algorithmen.

Wir bauen uns in den nächsten Abschnitten ein System von Funktionen auf, mit Hilfe derer wir jeden Algorithmus beschreiben können. Diese Funktionen haben jeweils $k \in \mathbb{N}$ natürliche Eingabewerte $n := n_1, \dots, n_k$, sind kompositional zusammensetzbar und verwenden Rekursion. Dabei definieren wir uns zunächst einen Minimalatz von Funktionen, die wir später gleich Bausteinen zu komplexeren Berechnungen zusammensetzen werden.

Mit diesen Vorbemerkungen können wir den intuitive (und wohl naive) Vorstellung der Berechenbarkeit präzisieren:

Definition:

Wir betrachten partielle Funktionen $f : \mathbb{N}^k \rightarrow \mathbb{N}$ und nennen diese **intuitiv berechenbar**, wenn es einen Algorithmus (eine Rechenvorschrift) gibt, der

- als Eingabe ein beliebiges k -Tupel (n_1, \dots, n_k) natürlicher Zahlen erhält und

- mit der Ausgabe $f(n_1, \dots, n_k)$ terminiert, falls $f(n_1, \dots, n_k) \neq \perp$.

Für solche Tupel (n_1, \dots, n_k) , für die $f(n_1, \dots, n_k)$ undefiniert ist (, d.h. $f(n_1, \dots, n_k) = \perp$), soll keine Ausgabe bereitgestellt werden; entweder läuft der Algorithmus endlos ohne Ausgabe oder er hält ebenfalls ohne Ausgabe an. Wie üblich bezeichnen wir mit $\mathbb{N} := 0, 1, 2, 3, \dots$ die Menge der natürlichen Zahlen inklusive der 0.

1 Rekursive Funktionen

1.1 Nicht berechenbare Funktionen

Es gibt Probleme, die nicht vollautomatisch lösbar sind.

Beispiel:

Hierzu betrachte man die Funktionenschar $f_r : \mathbb{N} \rightarrow \mathbb{N}$, wobei r eine beliebige irrationale Zahl im Intervall $]0, 1[$ ist.

$$f_r(n) := \begin{cases} 1, & \text{falls } n \text{ Anfangsstück des Nachkommanteils von } r \text{ ist} \\ 0, & \text{sonst.} \end{cases}$$

Die Funktionen f_r sind für unterschiedliches r offenbar verschieden, also ist die Menge

$$\{f_r : r \in]0, 1[\text{ mit } r \text{ irrational}\}$$

überabzählbar. Andererseits ist die Menge aller durch ein Computerprogramm berechenbarer Funktionen $f : \mathbb{N} \rightarrow \mathbb{N}$ abzählbar, da jedes Programm durch einen endlichen Text niedergeschrieben werden kann. Die Anzahl aller endlichen Texte über einem festen endlichen Alphabet ist abzählbar. Insbesondere muss es also nicht berechenbare Funktionen f_r geben.

Die eben gemachte Beobachtung präzisieren wir nun im folgenden

SATZ 1.1: *Es gibt nicht berechenbare Funktionen $f : \mathbb{N}^k \rightarrow \{0, 1\}$.*

Beweis. Ist die Funktion f berechenbar, dann existiert ein Algorithmus A , der f berechnet. Dabei ist A ein Wort endlicher Länge über einem endlichen Alphabet Σ . Weiter sei konstatiert, dass es nur abzählbar unendlich viele Wörter endlicher Länge über einem endlichen Alphabet Σ gibt, d.h. es kann nur abzählbar unendlich viele Algorithmen über Σ geben. Gibt es aber nur abzählbar unendlich viele Algorithmen so kann es gemäß Definition nur abzählbar unendlich viele berechenbare Funktionen $f : \mathbb{N}^k \rightarrow \mathbb{N}$ geben.

Nun betrachten wir die Menge aller partiellen Funktionen $\{f \mid f : \mathbb{N}^k \rightarrow \{0, 1\}\}$ und zeigen, dass diese Menge überabzählbar ist, woraus folgt, dass es nicht berechenbare Funktionen geben muss.

Angenommen, die Menge $\{f \mid f : \mathbb{N}^k \rightarrow \{0, 1\}\}$ wäre abzählbar. Dazu dürfen wir o.B.d.A. $k := 1$ setzen. Sei sodann $f_0(n), f_1(n), \dots, f_i(n), \dots$ eine Aufzählung unterschiedlicher Funktionen f_i . Wir definieren

$$g : \mathbb{N} \rightarrow \mathbb{N} \quad \text{mit}$$

$$i \mapsto g(i) := \begin{cases} 1, & f_i(i) = 0 \\ 0, & f_i(i) = 1 \end{cases}$$

Es liegt also $g \in \{f \mid f : \mathbb{N}^k \rightarrow \{0, 1\}\}$. Doch gemäß Definition kann g nicht in der obigen Aufzählung vorkommen, denn $g(i)$ unterscheidet sich im Vergleich zu f_i im Wert $f_i(i) \neq g(i)$, was zum Widerspruch führt. \square

Als Fazit halten wir also fest, dass es beweisbar nicht berechenbare Funktionen gibt!

1.2 Primitiv rekursive Funktionen

Die einfachste Klasse von rekursiven Funktionen sind die primitiv rekursiven Funktionen. Sie umfassen drei Arten von Funktionen und zwei Methoden, sie zu komplexeren Funktionen zusammensetzen.

Definition:

Die Klasse \mathcal{P} der **primitiv rekursiven Funktionen** ist die kleinste Klasse, welche die Basis-Funktionen (auch Grundfunktionen genannt)

- **Konstante Funktion**, $C^k : \mathbb{N}^k \rightarrow \mathbb{N}$ mit $(n_1, \dots, n_k) \mapsto c$ und $c \in \mathbb{N}$,
- **Nachfolgerfunktion** (Successor), $suc : \mathbb{N} \rightarrow \mathbb{N}$ mit $n \mapsto suc(n) := n + 1$
- **Projektion** (Auswahl) des j -ten Eintrags $\pi_j : \mathbb{N}^k \rightarrow \mathbb{N}$ mit $(n_1, \dots, n_j, \dots, n_k) \mapsto n_j$ und $1 \leq j \leq k$

enthält und abgeschlossen ist gegen

- **Substitution** (simultanes Einsetzen):
Sind $g : \mathbb{N}^k \rightarrow \mathbb{N}$ sowie $h_1 : \mathbb{N}^r \rightarrow \mathbb{N}, \dots, h_k : \mathbb{N}^r \rightarrow \mathbb{N}$ mit $1 \leq r \leq k$ primitiv rekursiv, so auch $f : \mathbb{N}^k \rightarrow \mathbb{N}$ mit $f(n_1, \dots, n_k) = g(h_1(n_1, \dots, n_r), \dots, h_k(n_1, \dots, n_r))$.

- **primitive Rekursion:**

Jede Funktion, die durch so genannte primitive Rekursion aus primitiv rekursiven Funktionen entsteht, ist primitiv rekursiv. Genauer: sind $g : \mathbb{N}^k \rightarrow \mathbb{N}$ und $h : \mathbb{N}^{k+2} \rightarrow \mathbb{N}$ primitiv rekursiv, so auch $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ mit

$$f(n_1, \dots, n_k, 0) = g(n_1, \dots, n_k)$$

$$f(n_1, \dots, n_k, m + 1) = h(n_1, \dots, n_k, m, f(n_1, \dots, n_k, m))$$

für $m \geq 0$.

Wir nennen eine Funktion f primitiv rekursiv, wenn $f \in \mathcal{P}$.

Beachten Sie, dass wir gemäß Vereinbarung das k -Tupel (n_1, \dots, n_k) auch abkürzend als n notieren können. Die Definition der primitiven Rekursion gleicht dem Induktionsprinzip für natürliche Zahlen: Ist der letzte Parameter von f echt größer 0, so wird die Teilfunktion h verwendet, um den Funktionswert $f(n, m + 1)$ zu berechnen, und zwar in Abhängigkeit von $f(n, m)$. Die Verankerung bildet dann $f(n, 0)$.

Es ist einsichtig, dass alle primitiv rekursiven Funktionen (also die Menge \mathcal{P}) intuitiv berechenbar sind, denn die Basisfunktionen sind berechenbar und sowohl die Komposition als auch die primitive Rekursion erzeugen aus berechenbaren Funktionen wieder solche.

Die Projektion π_j kann bspw. durch ein Array A berechnet werden, wobei man als Rückgabe gerade $A[i]$ erhält. Die Berechnung einer festen konstanten Funktion sollte ebenfalls keine Schwierigkeiten bereiten und die Nachfolgerfunktion $suc(n)$ erhält man durch simple Addition $n + 1$ den Rückgabewert.

Nun zeigen wir noch durch Angabe zweier Algorithmen, dass sowohl die Komposition als auch die primitive Rekursion berechenbar sind.

SUBSTITUTION:

Function f (N: array[1 ... k] of cardinal): cardinal;

```
var NN: array[1 ... r] of cardinal;  
var i: cardinal  
begin  
  NN[1]:=  $h(1, N)$ ;  
  ...  
  NN[k]:=  $h(k, N)$ ;  
   $f := g(NN)$ ;  
end
```

Der jeweilige Arrayeintrag $NN[i]$ speichert offensichtlich den Funktionswert von h_i . In der letzten Anweisungszeile wird dann, nachdem alle Funktionswerte von h_i mit $1 \leq i \leq k$ berechnet und gespeichert wurden, die Identität $f(n_1, \dots, n_k) = g(h_1(n_1, \dots, n_r), \dots, h_i(n_1, \dots, n_r), \dots, h_k(n_1, \dots, n_r))$ ausgenutzt. Diese muss gemäß Voraussetzungen bestehen.

PRIMITIVE REKURSION:

Function f (\mathbb{N} : array[1 ... (k+1)] of cardinal): cardinal;

```

var hh, i: cardinal;
begin
  hh:=  $g(N[1], N[2], \dots, N[k])$ ;
  i:=0;
  LOOP  $N[k+1]$  DO
    begin
      hh:= $h(N[1], \dots, N[k], i, hh)$ ;
      i:= i+1;
    end
  f:=hh;
end

```

Im nächsten Abschnitt bauen wir aus den Grundfunktionen komplexere auf, indem wir sie mittels der Substitution und primitiver Rekursion verknüpfen.

1.3 Arithmetische Funktionen, primitiv rekursiv ausgedrückt

Wir zeigen nun, dass die grundlegenden arithmetischen Operationen Addition, Subtraktion, Multiplikation und Division allesamt in \mathcal{P} liegen und sich damit primitiv rekursiv ausdrücken lassen. Die dadurch neu gewonnenen Funktionen können wir für den Aufbau weiterer Funktionen aus \mathcal{P} einsetzen.

Wie wir allerdings wissen bildet die Menge der natürlichen Zahlen \mathbb{N} zusammen mit der üblichen Zahlen-Addition keine Gruppe, da letztlich die additiv Inversen für die meisten Werte aus \mathbb{N} fehlen. Das heißt wir müssen insbesondere bei der Subtraktion darauf achten, dass wir uns stets innerhalb der Menge \mathbb{N} befinden. Dies bewerkstelligen wir durch eine modifizierte Definition dieser beiden Operationen.

Definition:

Es sei $Sub : \mathbb{N}^2 \rightarrow \mathbb{N}$ definiert durch

$$Sub(n, m) := \begin{cases} 0, & \text{falls } m \geq n \\ n - m, & \text{falls } m < n. \end{cases}$$

Wir nennen Sub die (allgemeine) **modifizierte Subtraktion** und sodann ist \mathbb{N} abgeschlossen gegebenüber Sub .

Im weiteren bezeichnen wir mit $Add(n, m) := n+m$ die übliche Addition, $Mult(n, m) := n \cdot m$, $Pot(n, m) := n^m$ die Potenzierung zweier natürlicher Zahlen. Außerdem soll $Fak(n) := n!$ die Fakultät einer natürlichen Zahl berechnen. Oftmals notieren wir $Add(n, m)$ auch abkürzend mit $n + m$; analoges soll für ähnlich elementare Funktionen gelten. Die modifizierte Subtraktion notieren wir abkürzend mit $Sub(n, m) = n - m$.

Lemma 1.2: *Es gilt*

- Die Identität id ,
- die Addition Add ,
- die Multiplikation $Mult$,
- die Potenzfunktion Pot ,
- die Fakultätsfunktion Fak sowie
- die modifizierte Subtraktion

liegen in \mathcal{P} , d.h. diese Funktionen sind allesamt primitiv rekursiv.

Beweis. Die Identität kann als Projektion einer einstelligen Funktion $id : \mathbb{N} \rightarrow \mathbb{N}$ mit $id(n) := n = \pi_1(n)$ dargestellt werden. Damit gilt $id \in \mathcal{P}$.

Die Additionsfunktion $Add : \mathbb{N}^2 \rightarrow \mathbb{N}$ ist primitiv rekursiv, denn sie kann primitiv rekursiv dargestellt werden in der Form

$$\begin{aligned} Add(n, 0) &= id(n) \\ Add(n, m + 1) &= suc(\pi_3(n, m + 1, Add(n, m))). \end{aligned}$$

Die logische Verbindung zwischen $Add(n, m + 1)$ und $Add(n, m)$ wird mit der Nachfolgerfunktion hergestellt. Die Funktion h besteht allerdings aus der Zusammensetzung der Nachfolgerfunktion suc und der Projektion π_3 . Dies ist notwendig, da lediglich auf das dritte Argument von $h(n, m + 1, Add(n, m))$ zugegriffen werden muss.

Da wir bereits wissen, dass $id, suc \in \mathcal{P}$ gilt und \mathcal{P} gegenüber der primitiven Rekursion abgeschlossen ist, folgt $Add \in \mathcal{P}$.

Genauso verfahren wir mit der Multiplikation $Mult : \mathbb{N}^2 \rightarrow \mathbb{N}$. Diese lässt sich primitiv rekursiv darstellen durch

$$\begin{aligned} Mult(n, 0) &= 0 \quad (\text{konstante Nullfunktion}) \\ Mult(n, m + 1) &= Add(\pi_1[n, m, Mult(n, m)], \pi_3[n, m, Mult(n, m)]). \end{aligned}$$

Gemäß Definition ist $g(n) := \widehat{0}$ die konstante Nullfunktion und damit $g \in \mathcal{P}$. Wieder muss die für die primitive Rekursion notwendige Funktion h die drei Argumente $(n, m, Mult(n, m))$ enthalten. Dies erfüllt obige Funktion offensichtlich und da wir von allen verwendeten Funktionen wissen, dass diese in \mathcal{P} liegen, folgt damit $Mult \in \mathcal{P}$.

Aufbauend auf der Multiplikation lässt sich die m -te Potenz von n , also $Pot(n, m)$, wie folgt darstellen:

$$\begin{aligned} Pot(n, 0) &= 1 && \text{(konstante Einsfunktion)} \\ Pot(n, m + 1) &= Mult(\pi_1[n, m, Pot(n, m)], \pi_3[n, m, Pot(n, m)]). \end{aligned}$$

Da sowohl $Mult$ als auch die konstante Einsfunktion in \mathcal{P} liegt, folgt $Pot \in \mathcal{P}$.

Die Fakultätsfunktion ist eine einstellige Funktion, dies sollte bei der rekursiven Darstellung beachtet werden:

$$\begin{aligned} Fak(0) &= 1 && \text{(konstante Einsfunktion)} \\ Fak(m + 1) &= Mult(\pi_1[m, Fak(m)], \pi_2[m, Fak(m)]). \end{aligned}$$

Es folgt also $Fak \in \mathcal{P}$.

Abschließend zeigen wir noch, dass die modifizierte Subtraktion in \mathcal{P} liegt. Dazu stellen wir Sub mit Hilfe der primitive Rekursion dar. Allerdings benötigen wir dazu noch die so genannte Vorgängerfunktion $Pre(n) := Sub(n, 1)$:

$$\begin{aligned} Pre(0) &= 0 && \text{(konstante Nullfunktion)} \\ Pre(m + 1) &= \pi_2[m, Pre(m)]. \end{aligned}$$

Da $\pi_i, C \in \mathcal{P}$ gilt, folgt $Pre \in \mathcal{P}$. Nun kommen wir sogleich zur primitiv rekursiven Darstellung von Sub :

$$\begin{aligned} Sub(n, 0) &= \pi_1(n) \\ Sub(n, m + 1) &= Pre(\pi_3[n, m, Sub(n, m)]) \end{aligned}$$

Damit ist bereits $Sub \in \mathcal{P}$ gezeigt. □

Mit Hilfe der elementaren arithmetischen Funktionen können wir rekursive Funktionen schon etwas bequemer beschreiben, als wenn wir nur die Basis-Funktionen dafür verwenden würden.

In der Definition der Komposition (simultanes Einsetzen) sind alle Funktionen h_i mit $1 \leq i \leq k$ von denselben Argumenten (n_1, \dots, n_r) abhängig, wobei $1 \leq r \leq k$ gilt. Nun

könnte es allerdings sein, dass manche der h_i nicht alle Argumente oder aber die vorhandenen Argumente in einer differenten Reihenfolge benötigen. Das all dies im Rahmen von \mathcal{P} erlaubt ist bringt folgendes Lemma zum Ausdruck.

Lemma 1.3: *Die Funktionenmenge \mathcal{P} ist gegen Umordnen und Weglassen von Variablen bezüglich der Komposition abgeschlossen.*

Beweis. Die Menge der primitiv rekursiven Funktionen ist abgeschlossen gegenüber Umordnung der Argumente. Sei $n := (n_1, \dots, n_k)$ unser Ausgangstupel mit k Positionen, f eine primitiv rekursive Funktion und $m := (m_1, \dots, m_k)$ das durch Umordnung entstandene Tupel mit k Einträgen, d.h., die Einträge von m ergeben sich durch Permutation von n in k Worten. Da jede Permutation bijektiv ist, muss die entsprechende inverse Permutation π existieren. Die Permutation π kann mit Hilfe von k Projektionen $\pi_{j_1}, \dots, \pi_{j_k}$ mit $1 \leq j_i \leq k$ für $1 \leq i \leq k$ und der *Komposition* dargestellt werden. Wie wir wissen sind Projektionen primitiv rekursiv. Also ergibt sich sodann die Darstellung:

$$f(n) = f(\pi_{j_1}(m_1), \dots, \pi_{j_k}(m_k))$$

Auch das Weglassen von Argumenten, die nicht benötigt werden stört diese Argumentation nicht, da diese Argumente dann einfach festgelassen werden. Es reduziert sich dann lediglich die Anzahl der benötigten Projektionen. \square

Ein Beispiel wird den letzten Satz veranschaulichen.

Beispiel:

Es sei $f(v, w, x, y)$ eine primitiv rekursive Funktion in vier Argumenten. Besteht die Identität $g(a, b, c) = f(b, b, c, a)$, dann ist auch die Funktion g primitiv rekursiv. D.h. man kann sämtliche Bilder von f offensichtlich auch mit Hilfe von drei Argumenten beschreiben, wie dies g ja gerade zeigt. Eine formalen Beweis kann man mit Hilfe der Komposition und der Projektion erbringen:

$$g(a, b, c) = f(\pi_2(a, b, c), \pi_2(a, b, c), \pi_3(a, b, c), \pi_1(a, b, c)).$$

Eine häufig verwendete Konstruktion bei der Definition von Funktionen ist die Fallunterscheidung. Wenn man die Typen von Fällen, die untersucht werden können, geeignet einschränkt, kann man auch dieses Konstrukt mit den Mitteln der primitiv rekursiven Funktionen ausdrücken.

Lemma 1.4: *Die Funktionenmenge \mathcal{P} ist abgeschlossen gegen primitiv rekursive Fallunterscheidung, d.h. wenn g_i, h_i für $1 \leq i \leq k$ primitiv rekursiv sind, und es für alle*

$n = (n_1, \dots, n_k)$ genau ein j gibt, so dass $h_j(n) = 0$ gilt, so ist auch folgende Funktion f primitiv rekursiv:

$$f(n) := \begin{cases} g_1(n), & \text{falls } h_1(n) = 0 \\ \dots, & \dots \\ g_k(n), & \text{falls } h_k(n) = 0 \end{cases}$$

Beweis. Die Funktion f können wir mit Hilfe der modifizierten Subtraktion $\dot{-}$ darstellen durch

$$f(n) = g_1(n) \cdot [1 \dot{-} h_1(n)] + \dots + g_k(n) \cdot [1 \dot{-} h_k(n)]$$

und für die verwendeten Teilfunktionen $\cdot, +, \dot{-}$ haben wir bereits bewiesen, dass sie in \mathcal{P} liegen. \square

Der eben geführte Beweis hätte auch sehr schön mit Hilfe der charakteristischen Funktion erbracht werden können.

1.4 Primitiv rekursive Relationen

Die Technik, die der letzte Beweis von Lemma 1.4 verwendet, verdient genauere Betrachtung. Fallunterscheidung wird ausgedrückt mit Hilfe der primitiv rekursiven Funktionen $\cdot, +$ und $\dot{-}$, die hier die Aufgabe logischer Konnektoren übernehmen:

- $n \cdot m \neq 0 \Leftrightarrow n \neq 0$ **und** $m \neq 0$.
- $n + m \neq 0 \Leftrightarrow n \neq 0$ **oder** $m \neq 0$.
- $1 \dot{-} n \neq 0 \Leftrightarrow n$ ist **nicht** ungleich 0.

Für die weiteren Beweise benötigen wir noch den Nachweis, dass $\max(n, m), \min(n, m)$ sowie

$$sg(n) := \begin{cases} 1, & \text{falls } n > 0 \\ 0, & \text{falls } n = 0. \end{cases}$$

primitiv rekursive Funktionen sind. Wir können das Maximum als Komposition primitiv rekursiver Funktionen darstellen.

$$\max(n, m) = \text{Add}[\pi_1(n, m), \text{Sub}(\pi_2(n, m), \pi_1(n, m))]$$

Entsprechend lässt sich auch die Funktion $\min(n, m)$ als Komposition primitiv rekursiver Funktionen darstellen.

$$\min(m, n) = \text{Sub}[\pi_2(n, m), \text{Sub}\{\pi_2(n, m), \pi_1(n, m)\}]$$

Da $\text{Sub}, \pi_i, \text{Add} \in \mathcal{P}$ folgt damit, dass $\min, \max \in \mathcal{P}$. Offensichtlich ist auch die Signum-Funktion primitiv rekursiv, da gilt

$$\begin{aligned} \text{sgn}(0) &= 0 && \text{(konstante Nullfunktion)} \\ \text{sgn}(m+1) &= 1 && \text{(konstante Einsfunktion).} \end{aligned}$$

Bemerkung 1:

Künftig werden wir verstärkt Gebrauch der abkürzenden Schreibweise der bereits bekannten primitiv rekursiven Funktionen machen.

Prädikate, die aus Vergleichsoperationen sowie den Operationen Negation, Disjunktion und Konjunktion aufgebaut sind, sind durch primitiv rekursive Funktionen darstellbar. Solche Prädikate werden deshalb ebenfalls primitiv rekursiv genannt:

Definition:

Eine **Relation** $R \subseteq \mathbb{N}^k$ heißt **primitiv rekursiv**, falls ihre charakteristische Funktion χ_R primitiv rekursiv ist.

Bemerkung 2:

- Beachten Sie, dass man mit Hilfe von Relationen auch Prädikate vollständig charakterisieren kann. Ein dreistelliges Prädikat $\alpha = X_1 \vee \neg X_2 \wedge X_3$ gibt bspw. Beziehungen zwischen drei Subjekten, hier den Literalen, an. Wie wir wissen, können Beziehungen mit Hilfe von Relationen formuliert werden.
- Relationen erklären in natürlicher Art und Weise Teilmengenbeziehungen. So ist es nicht verwunderlich, dass man auch primitiv rekursive Mengen definieren kann.

Beispiel:

Wir zeigen nun, dass die Relationen $<, \leq$ und $=$ primitiv rekursiv sind.

Die Gleichheitsrelation $=$ erhalten wir durch $\chi_=(x, y) = 1 - [(x - y) + (y - x)]$ - diese Funktion ist genau dann 0, wenn $x = y$ gilt. Da sich $\chi_ =$ aus primitiv rekursiven Funktionen zusammensetzt folgt damit, dass $=$ -Relation primitiv rekursiv ist.

Die Kleinerrelation $n < m$ erhalten wir mit Hilfe der Signum-Funktion und der modifizierten Subtraktion durch: $\chi_<(n, m) := \text{sgn}(m - n)$. Dies wird klar, wenn man sich

folgende Äquivalenzkette betrachtet: $n < m \Leftrightarrow m - n > 0 \Leftrightarrow \text{sgn}(m - n) = 1$.

Da die Addition eine primitiv rekursive Operation darstellt, ergibt sich die Eigenschaft „primitiv rekursiv“ der Relation \leq (Veroderung von $=$ und $<$) durch $\chi_{<} + \chi_{=}$. Zu beachten ist, dass sowohl $\chi_{<}$ als auch $\chi_{=}$ primitiv rekursiv sind und deren Summe genau dann 1 ergibt, wenn $\chi_{<}$ oder $\chi_{=}$ gleich 1 ist, d.h., wenn gilt $\chi_{<} = 1 \vee \chi_{=} = 1$.

Folgendes Lemma gibt einen ersten Einblick in die Struktur des Raumes \mathcal{P} :

Lemma 1.5: *Es seien die Prädikate $p, q \subseteq \mathbb{N}^r$ primitiv rekursiv, d.h. $p, q \in \mathcal{P}$. Dann gilt*

1. *Das Prädikat $p \cup q \subseteq \mathbb{N}^r$ ist primitiv rekursiv.*
2. *Das Prädikat $p \cap q \subseteq \mathbb{N}^r$ ist primitiv rekursiv.*
3. *Das Prädikat $\neg p \subseteq \mathbb{N}^r$ und $\neg q \subseteq \mathbb{N}^r$ sind primitiv rekursiv.*

Beweis. Zuerst zeigen wir den ersten Punkt.

Es sei $x := (x_1, \dots, x_r) \in \mathbb{N}^r$, dann ist offensichtlich

$$\begin{aligned} \chi_{p \cup q}(x) &= \begin{cases} 1, & \text{falls } x \in p \vee x \in q \\ 0, & \text{falls } x \notin p \wedge x \notin q \end{cases} \\ &= \text{sgn}(\chi_p(x) + \chi_q(x)). \end{aligned}$$

Da die Funktion $\chi_{p \cup q}$ aus Substitution bereits bekannter primitiv rekursiver Funktionen $+$, sgn und der charakteristischen Funktion χ_A einer Menge A hervorgeht, folgt die Behauptung.

Es ist diesmal

$$\begin{aligned} \chi_{p \cap q}(x) &= \begin{cases} 1, & \text{falls } x \in p \wedge x \in q \\ 0, & \text{falls } x \notin p \vee x \notin q \end{cases} \\ &= \text{sgn}(\chi_p(x) \cdot \chi_q(x)). \end{aligned}$$

Da die Funktion $\chi_{p \cap q}$ aus Substitution bereits bekannter primitiv rekursiver Funktionen \cdot , sgn , χ_A hervorgeht, folgt die Behauptung.

Wieder definieren wir (für p)

$$\begin{aligned}\chi_{\neg p}(x) &= \begin{cases} 1, & \text{falls } x \notin p \\ 0, & \text{falls } x \in p \end{cases} \\ &= 1 \dot{-} \chi_p(x).\end{aligned}$$

Da die Funktion $\chi_{\neg p}$ aus Substitution bereits bekannter primitiv rekursiver Funktionen $C, \dot{-}, \chi_A$ hervorgeht, folgt die Behauptung für p . Analog zeigt man die Behauptung für q .

□

Abschließend betrachten wir noch ein typisches Beispiel in diesem Kontext.

Beispiel:

Es sei $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ eine primitiv rekursive Funktion. Wir zeigen, dass das Prädikat $p := \{(x, y, f(x, y)) \mid (x, y) \in \mathbb{N}^2\}$ primitiv rekursiv ist. Dazu müssen wir nachweisen, dass die charakteristische Funktion χ_p primitiv rekursiv ist.

$$\begin{aligned}\chi_p(x, y, z) &= \begin{cases} 1, & \text{falls } f(x, y) - z = 0 \\ 0, & \text{sonst} \end{cases} \\ &= 1 \dot{-} \text{sgn}(|z - f(x, y)|).\end{aligned}$$

Da sich also die charakteristische Funktion aus Substitutionen bereits bekannter primitiv rekursiver Funktionen zusammensetzt, folgt die Behauptung.

1.5 Der μ -Operator

Da die primitiv rekursiven Funktionen \mathcal{P} nur eine echte Teilmenge der berechenbaren Funktionen sind, muss man, um alle berechenbaren Funktionen in mathematischer Notation zu beschreiben, die Klasse \mathcal{P} *echt* erweitern.

Dies kann man dadurch versuchen, indem man die Klasse der elementaren Basis-Funktionen erweitert, und/oder zum anderen, indem neben der Substitution und der Rekursion weitere Konstruktionsprinzipien zugelassen werden. Es zeigt sich, dass ein weiteres Konstruktionsprinzip ausreicht, um *alle* berechenbaren Funktionen $\mathcal{F}_{\text{berech}}$ zu erfassen.

1.5.1 Der unbeschränkte μ -Operator

Definition:

Es sei $g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ eine totale primitiv-rekursive Funktion mit

$$g(n_1, \dots, n_k, y) = g(n, y).$$

Dann heißt die Funktion

$$f(n) = \mu_y[g(n, y)]$$

die **Minimalisierung** von g bezüglich y , wenn gilt

$$f(n) := \begin{cases} y, & \text{falls } g(n, y) = 0 \text{ und } g(n, z) \neq 0 \text{ für alle } z < y, \\ \perp, & \text{falls } g(n, y) \neq 0 \text{ für alle } y \in \mathbb{N}. \end{cases}$$

Man nennt μ den **unbeschränkten μ -Operator**.

Da g eine totale Funktion ist, ist $g(n, y)$ für alle (n, y) definiert. Wie mächtig die Minimalisierung für die Funktionsbildung ist, zeigen die folgenden Beispiele.

Beispiel:

a) $f(x) := \mu_y[x \bmod 2 + y]$ liefert

$$f(x) = \begin{cases} 0, & \text{für gerades } x \\ \perp, & \text{für ungerades } x \end{cases}$$

Ist also $x = 2t$ mit $t \in \mathbb{N}$ und damit gerade, so gilt $x \bmod 2 = 0$. Damit kann man $y = 0$ wählen, so dass insgesamt $x \bmod 2 + y = 0$ gilt. Ist dagegen x ungerade, also $x = 2t + 1$ für ein $t \in \mathbb{N}$, so ist $x \bmod 2 \neq 0$ und damit auch $x \bmod 2 + y \neq 0$. Deshalb existiert kein y , welches den Anforderungen genügt und somit folgt $f(x) = \perp$.

b) $f(x) := \mu_y[x^2 \dot{-} y]$ liefert $f(x) = x^2$. Schließlich muss minimales $y \in \mathbb{N}$ gewählt werden, so dass $x^2 \dot{-} y = 0 \Leftrightarrow x^2 = y$.

Setzen wir dagegen $f(x) := \mu_y[y^2 \dot{-} x]$ so ergibt sich die Minimalisierung $f(x) = 0$, da $0^2 \dot{-} x$ für beliebiges $x \in \mathbb{N}$ stets den Wert 0 ergibt. Dazu beachte man die Definition der modifizierten Subtraktion.

c) Sei p ein Prädikat mit

$$p(x, y) := \begin{cases} 1, & \text{falls } y \text{ eine Primzahl größer als } x \text{ ist,} \\ 0, & \text{sonst.} \end{cases}$$

Die Funktion

$$f(x) = \mu_y[1 \dot{-} p(x, y)]$$

liefert die kleinste Primzahl, die größer als x ist.

Die durch Anwendung des μ -Operators auf g entstehende Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ gibt das *kleinste* y mit $g(n, y) = 0$ wieder, falls dieses überhaupt existiert. Existiert kein kleinstes Element, so dass $g(n, y) = 0$ gilt, wir also $\min\{\emptyset\}$ bestimmen müssen, so soll $\min\{\emptyset\} = \perp$ ergeben, also undefiniert sein.

Das Adjektiv „undefiniert“ sollte einen in der theoretischen Informatik stets an die so genannten *partitellen* (d.h. nicht totalen) Funktionen erinnern. Das heißt, durch Anwendung des μ -Operators können wirklich partielle Funktionen entstehen. So erklärt sich auch die hin und wieder verwendeten Bezeichnungen *partiell-rekursiv* und *total-rekursiv*. Erstere sind partielle μ -rekursive Funktionen und letztere total μ -rekursive Funktionen.

Beispielsweise entsteht durch Anwendung des μ -Operators auf die zweistellige, konstante Funktion $g(x, y) = 1$ die vollständig undefinierte Funktion Ω . Ist $\mathcal{F}_{part} := \{f \mid f \text{ ist partiell-rekursiv}\}$ die Klasse der partiell μ -rekursiven Funktionen, so ist \mathcal{F}_{part} eine echte Obermenge der primitiv-rekursiven Funktionen. Da auch die partiell rekursiven Funktionen berechenbar sind muss \mathcal{F}_{part} eine Teilmenge aller berechenbaren Funktionen sein.

Definition:

Die Klasse \mathcal{F}_μ der **μ -rekursiven Funktionen** ist die kleinste Klasse von (evtl. partiellen) Funktionen, die die Basisfunktion enthält und abgeschlossen ist unter Substitution, primitiver Rekursion und Anwendung des μ -Operators.

Entsteht $f(n)$ durch Minimalisierung von $g(n, y)$, so ist $f(n)$ berechenbar, falls es ein y mit $g(n, y) = 0$ gibt. Ist nämlich

Function $g(\mathbb{N}: \text{array } [1 \dots k] \text{ of cardinal}): \text{cardinal}$

eine Funktionsprozedur zur Berechnung von g , dann leistet die Prozedur

```
Function  $f(\mathbb{N}: \text{array } [1 \dots k] \text{ of cardinal}): \text{cardinal}$ 
var y:cardinal;
begin
```

```

y:= 0;
WHILE g(N,y) ≠ 0 Do y:= y +1;
f:=y;

end

```

das Gewünschte.

Ist jedoch $g(n, y) \neq 0$ für alle $y \geq 0$, so terminiert die Funktionsprozedur **Function** f nicht, also ist die Funktion f nicht berechenbar für das Argument n .

Beispiel:

Wir zeigen, dass die Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ definiert durch $f(x) := \lceil \sqrt{x} \rceil$ μ -rekursiv ist.

Dazu müssen wir nachweisen, dass man f , mit Hilfe von primitiv rekursiven Funktionen und des μ -Operators, erzeugen kann. Um einen Hinweis auf den Lösungsweg dieser oder gleichartiger Aufgaben zu bekommen, bietet es sich an, die Definition der zu untersuchenden Funktion (hier f) näher zu betrachten: Die Quadratwurzel \sqrt{x} der Zahl x ist die kleinste positive Zahl $z \in \mathbb{N}$, so dass die Gleichung $z^2 = x$ erfüllt ist. Entsprechend ergibt sich

$$\begin{aligned} \lceil \sqrt{x} \rceil &= \min\{z \geq 0 \mid z^2 \geq x\} \\ &= \min\{z \geq 0 \mid x - z^2 = 0\} \end{aligned}$$

und somit entsteht die Funktion f durch Anwendung des μ -Operators auf f bezüglich $x - z^2$. Da wir gemäß Lemma 1.2 wissen, dass die modifizierte Differenz und die Multiplikation primitiv rekursiv sind, folgt damit bereits die Behauptung.

1.5.2 Der beschränkte μ -Operator

Wir wissen nun also, dass der unbeschränkte μ -Operator als „funktionales Äquivalent“ einer WHILE-Schleife interpretiert werden kann. Genau aus diesem Grund ist dieser Operator abgeschlossen bezüglich der Menge der berechenbaren Funktionen.

Es existiert aber auch eine eingeschränkte Version des μ -Operators - eben der beschränkte μ Operator, welcher als „funktionales Äquivalent“ einer FOR-Schleife angesehen werden kann.

Definition:

Sei $g : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ eine Funktion, dann ist der **beschränkte μ -Operator** wie folgt definiert:

$$\mu_{x < z}[g(n, y)] := \begin{cases} y, & \text{falls } g(n, y) = 0 \text{ und } \forall x < y : g(n, x) \neq 0 \\ & \text{und } 0 \leq y < z \\ 0, & \text{falls } g(n, x) \neq 0 \forall x \text{ mit } 0 \leq x < z \\ & \text{oder } z = 0 \end{cases}$$

An der Definition können wir ablesen: Ergibt die Anwendung des beschränkten μ -Operators den Wert 0, so kann das zweierlei Gründe haben: Entweder nimmt für y den Wert 0 an (erster Fall der Definition), oder die Funktion g nimmt für kein Argument $x < z$ den Wert 0 an (zweiter Fall der Definition).

Im Gegensatz zum unbeschränkten μ -Operator operiert der beschränkte μ -Operator abgeschlossen auf der Menge der primitiv rekursiven Funktionen \mathcal{P} . Wenden wir also den beschränkten μ -Operator auf eine beliebige primitiv rekursive Funktion an, so ist das Ergebnis dieser Operation wieder eine primitiv rekursive Funktion. Ein Beweis ist mit den bereits gewonnenen Erkenntnissen nicht schwer: Wir wissen, dass primitiv rekursive Funktionen gegenüber Fallunterscheidungen und der Verknüpfung mit logischen Konnektoren invariant sind.

Das auch der beschränkte μ -Operator berechenbar ist liegt auf der Hand:

```
Function  $f$ (N: array [1 .. k+1] of cardinal): cardinal
var y:cardinal;
begin
    FOR y:=0 TO z DO;
    begin
        IF  $g(n, y) = 0$  THEN return y;
        ELSE y:= y+1;
    end
    return 0;
end
```

Abschließend untersuchen wir im folgenden Abschnitt die so genannte Ackermann-Funktion, eine Funktion die nicht primitiv rekursiv ist, jedoch berechenbar. Wir werden zeigen, dass die Ackermann-Funktion stärker wächst als jede primitiv rekursive Funktion.

2 Die Ackermann-Funktion

Zunächst definieren wir die Ackermannfunktion rekursiv:

Definition:

Es sei $A : \mathbb{N}_0^2 \rightarrow \mathbb{N}$ eine Funktion mit der rekursiven Funktionsvorschrift

$$(Ack_1) \quad A(0, y) := y + 1,$$

$$(Ack_2) \quad A(x, 0) := A(x - 1, 1) \text{ für } x > 0,$$

$$(Ack_3) \quad A(x, y) := A(x - 1, A(x, y - 1)) \text{ für } x > 0, y > 0.$$

Die Funktion A nennen wir **Ackermann-Funktion**.

Auf Grundlage dieser Definition beweisen wir einige Hilfssätze die wir später benötigen werden, um zu zeigen, dass die Ackermann-Funktion stärker wächst als jede primitiv rekursive Funktion.

Lemma 2.1: *Die Ackermannfunktion ist wertmäßig größer als ihr zweites Argument angibt, also gilt:*

$$A(x, y) > y \text{ für alle } x, y \in \mathbb{N}_0$$

Beweis. (durch Induktion nach x)

In diesem Beweis seien x, y stets wie in der obigen Behauptung definiert. Für $x = 0$ gilt $0 < A(0, y) = y + 1$ gemäß Definition, d.h. es gilt die Induktionsverankerung. Wir nehmen nun an, die Behauptung $y < A(x, y)$ gelte für ein beliebiges jedoch fixiertes x und alle y (Induktionsannahme für x).

Sodann müssen wir $y < A(x + 1, y)$ für alle y zeigen. Diesen Beweis führen wir mit Induktion nach der zweiten Variablen y , d.h. wir fahren nun mit Induktion nach y fort:

Nach Induktionsvoraussetzung für x gilt $1 < A(x, 1)$. Wir wollen nun die Induktionsverankerung für $y = 0$ zeigen, das also $0 < A(x + 1, 0)$ gilt. Gemäß Definition (Ack_2) ist $A(x + 1, 0) = A(x, 1)$ und wie wir wissen gilt $A(x, 1) > 1 > 0$. Somit haben wir also den Induktionsanfang für y bewiesen.

Nehmen wir nun an, die Behauptung gelte für $x + 1$ und y , d.h. es gilt $y < A(x + 1, y)$. Es ist dann zu zeigen, dass $y + 1 < A(x + 1, y + 1)$ folgt.

Wir setzen in die Induktionsannahme für x (also in $y < A(x, y)$) den speziellen Wert $A(x + 1, y)$ für das zweite Argument y ein und erhalten: $A(x + 1, y) < A(x, A(x + 1, y)) = A(x + 1, y + 1)$. Zusammen mit der Induktionsannahme für y - und dem vorher Gesagtem ergibt sich also $y + 1 < A(x + 1, y + 1)$. \square

Lemma 2.2: *Desweiteren wächst $A(x, y)$ streng monoton im y -Argument, also gilt:*

$$A(x, y) < A(x, y + 1) \text{ für } x \geq 0, y \geq 0.$$

Beweis. Für $x = 0$ ist $A(0, y) = y + 1 < y + 2 = A(0, y + 1)$. Für alle anderen x , d.h. für $x > 0$ folgt die Behauptung mit Hilfe des letzten Lemma [vgl. (1)]. Dazu substituieren wir $y' := A(x, y)$ und $x' := x - 1$ in (1).

$$A(x', y') > y' \text{ für alle } x', y' \in \mathbb{N}_0 \quad (2.1)$$

$$\Rightarrow A(x - 1, A(x, y)) > A(x, y). \quad (2.2)$$

Aus (3) ergibt sich gemäß Definition (Ack_2) der Ackermannfunktion $A(x, y) < A(x, y + 1)$. \square

Eine weitere wichtige Eigenschaft ist die im nun folgenden Lemma formulierte:

Lemma 2.3: *Die Ackermannfunktion $A(x, y)$ wächst im ersten Argument stärker als im zweiten Argument, also gilt:*

$$A(x, y + 1) \leq A(x + 1, y) \text{ für } x \geq 0, y \geq 3.$$

Beweis. (durch Induktion nach y)

Der Induktionsanfang für $y = 0$ gilt, da offensichtlich $A(x, 0 + 1) \leq A(x + 1, 0) = A(x, 1)$ gilt, wobei die letzte Identität durch (Ack_2) folgt.

Wir nehmen nun an, dass die Induktionsvoraussetzung $A(x, y + 1) \leq A(x + 1, y)$ für alle $x \geq 0$ und fixiertes $y \geq 3$. Nach Lemma 1 ist $y + 1 < A(x, y + 1)$, also gilt auch $y + 2 \leq A(x, y + 1) \leq A(x + 1, y)$ aufgrund der Induktionsannahme. Mit Lemma 2 und der eben gemachten Abschätzung, sowie mit (Ack_3) ergibt sich damit $A(x, y + 2) \leq A(x, A(x + 1, y)) = A(x + 1, y + 1)$ was zu zeigen war. \square

Analoges zu Lemma 2 gilt auch für das erste Argument der Ackermannfunktion:

Lemma 2.4: $A(x, y)$ wächst streng monoton im x -Argument, also gilt:

$$A(x, y) < A(x + 1, y).$$

Beweis. Mit Lemma 2 und Lemma 3 erhält man unmittelbar: $A(x, y) < A(x, y + 1) \leq A(x + 1, y)$. \square

Die eben bewiesenen Lemmata können dazu benutzt werden, um zu zeigen, dass es eine Konstante $k \in \mathbb{N}_0$ gibt, so dass

$$\sum_{i=1}^n A(k_i, y), \quad n \geq 1,$$

für beliebige Konstanten $k_i \in \mathbb{N}_0$ und alle $y > 2$ gilt.

Beweis. Der Fall $n = 1$ ist trivial, denn die Ackermann ist monoton steigend im ersten Argument, so ist also nur $k > k_0$ zu wählen.

Sei nun $n = 2$ und $k_{max} := \max\{k_1, k_2\}$ das Maximum der beiden Konstanten. Dann gilt:

$$\begin{aligned} A(k_1, y) + A(k_2, y) &\leq 2A(k_{max}, y) = A(1, A(k_{max}, y)) \\ &< A(k_{max} + 1, A(k_{max} + 2, y)) \\ &= A(k_{max} + 2, y + 1) \\ &< A(k, y) \end{aligned}$$

Setzen wir also nun $k := k_{max} + 3$ so folgt die Behauptung für diesen Fall.

Ist $n \geq 2$, so ergibt sich durch Induktion nach n :

$$\begin{aligned} \sum_{i=1}^{n+1} A(k_i, y) &= (A(k_1, y) + A(k_2, y)) + \sum_{i=3}^{n+1} A(k_i, y) \\ &< A(\hat{k}, y) + A(\tilde{k}, y) \quad (\text{gemäß Induktionsvoraussetzung}) \\ &< A(k, y) \quad (\text{gemäß Induktionsvoraussetzung}) \end{aligned}$$

\square

Die eben nachgewiesene Eigenschaft lässt vermuten, dass die Ackermannfunktion stärker wächst als jede andere primitiv-rekursive Funktion, wenn man nur das erste Argument hinreichend groß genug wählt.

Nun zeigen wir, dass es zu jeder primitiv rekursiven Funktion $f : \mathbb{N}_0^r \rightarrow \mathbb{N}_0^r$ ein $k \in \mathbb{N}_0$ gibt, derart dass

$$f(x_1, x_2, \dots, x_r) < A(k, x_1, x_2, \dots, x_r) \text{ mit } x_1 + x_2 + \dots + x_r > 2,$$

gilt. Um dies zu beweisen, genügt es, die Gültigkeit dieser Aussage für die elementaren Funktionen, die Substitution und die Rekursion nachzuweisen.

a) Nullfunktion $C_0^r(x_1, x_2, \dots, x_r) = 0 < A(0, 0) < A(0, x_1 + x_2 + \dots + x_r)$.

b) Nachfolgerfunktion $suc(x_1) = x_1 + 1 < A(0, x_1)$.

c) Projektionsfunktion $\pi_i^r(x_1, x_2, \dots, x_r) = x_i < A(0, x_1 + x_2 + \dots + x_r)$.

d) Substitution: Es sei $g(x_1, x_2, \dots, x_r) < A(0, x_1 + x_2 + \dots + x_r)$,

$$h_i(x_1, x_2, \dots, x_m) < A(k_i, x_1 + x_2 + \dots + x_m) \text{ für } 1 \leq i \leq r.$$

$$\begin{aligned} f(x_1, x_2, \dots, x_m) &= g(h_1, \dots, h_r) \\ &< A(k_0, h_1 + \dots + h_r) \\ &< A(k_0, A(c_1, x_1 + \dots + x_m) + \dots + A(c_r, x_1 + \dots + x_m)) \\ &< A(k_0, A(\hat{k}, x_1 + \dots + x_m)) \\ &< A(k_0 + \hat{k}, A(\hat{k} + k_0 + 1, x_1 + \dots + x_m)) \\ &= A(k_0 + \hat{k} + 1, x_1 + \dots + x_m) \\ &= A(k, x_1 + \dots + x_r). \end{aligned}$$

d) Rekursion:

Es seien $g(x_1, \dots, x_r) < A(0, x_1 + \dots + x_r)$ und $h_i(x_1, \dots, x_{r+2}) < A(k_1, x_1 + \dots + x_{r+2})$.

- $f(x_1, x_2, \dots, x_r, 0) = g(x_1, \dots, x_r) < A(k_0, x_1 + \dots + x_r + 0)$
- Induktionsannahme: $f(x_1, \dots, x_{r+1}, 0) < A(\hat{k}, x_1 + \dots + x_{r+1})$ für $n_{r+1} \geq 0$.
- Induktionsschritt:

$$\begin{aligned} f(x_1, \dots, x_{r+1} + 1) &= h(x_1, \dots, x_{r+1}, f(x_1, \dots, x_{r+1})) \\ &< A(k_1, x_1 + \dots + x_{r+1} + f(x_1, \dots, x_{r+1})) \\ &< A(k_1, x_1 + \dots + x_{r+1} + A(\hat{k}, x_1, \dots, x_{r+1})) \\ &< A(k_1, A(0, x_1 + \dots + x_{r+1}) + A(\hat{k}, x_1, \dots, x_{r+1})) \\ &\leq A(k_1, A(k_2, x_1 + \dots + x_{r+1})) \\ &< A(k_1 + k_2, A(k_1 + k_2 + 1, x_1 + \dots + x_{r+1})) \\ &= A(k_1 + k_2 + 2, x_1 + \dots + x_{r+1} + 1) \\ &< A(k_1 + k_2 + 2, x_1 + \dots + x_{r+1}) \\ &= A(k, x_1 + \dots + x_{r+1}) \end{aligned}$$

Weiterhin viel Spaß mit der Informatik!
<http://www.mathematik-netz.de> und
<http://www.mathering.de>