Rekursive Funktionen

Man kann eine Funktion $f : A \to B$ durch einen Term definieren, der selbst Aufrufe von f enthält.

Beispiel:

$$\text{fakultät} = \mathbf{function}(n)\textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \cdot \text{fakultät}(n-1)$$

Dies bezeichnet man als *rekursive Definition*.

Wie man formell den Wert einer rekursiv definierten Funktion (kurz: rekursiven Funktion) bestimmt, sehen wir später.

Jetzt rechnen wir einfach aus:

$$\text{fakultät}(3) = 3{\cdot}\text{fakultät}(2) = 3{\cdot}2{\cdot}\text{fakultät}(1) = 3{\cdot}2{\cdot}1{\cdot}\text{fakultät}(0) = 3{\cdot}2{\cdot}1{\cdot}1 = 6$$

Beachte: Rekursion ist eine Quelle von Undefiniertheit:

Wenn

$$f = \mathbf{function}(n)\mathbf{if}\ n = 0\ \mathbf{then}\ 1\ \mathbf{else}\ f(n+1)$$

dann ist $f(0) = 1$ und $f(n)$ undefiniert für $n > 0$. Also $f : \mathbb{N} \to \mathbb{N}$, aber $D(f) = \{0\}$.

Mehr Rekursion

Die Fibonacci-Zahlen:

$$\text{fib} : \mathbb{N} \to \mathbb{N}$$

$$\text{fib} = \textbf{function}(n)\textbf{if } n = 0 \textbf{ then } 1 \textbf{ else if } n = 1 \textbf{ then } 1 \textbf{ else } \text{fib}(n-1) + \text{fib}(n-2)$$

Interpretation: $\text{fib}(n) =$ Hasenpopulation nach n Monaten unter der Annahme, dass Hasen jeden Monat einen Nachkommen haben, dies aber erst ab dem zweiten Lebensmonat.

$$\text{fib}(n) \sim (\frac{\sqrt{5}+1}{2})^n$$

Nochmal 3n+1

$$g = \textbf{function}(n)\textbf{if } n \bmod 2 = 0 \textbf{ then } n/2 \textbf{ else } 3n+1$$

$$f = \textbf{function}(n)\textbf{if } n = 1 \textbf{ then } 0 \textbf{ else } 1 + f(g(n))$$

Türme von Hanoi

Es gibt drei senkrechte Stäbe. Auf dem ersten liegen n gelochte Scheiben von nach oben hin abnehmender Größe.

Man soll den ganzen Stapel auf den dritten Stab transferieren, darf aber immer nur jeweils eine Scheibe entweder nach ganz unten oder auf eine größere legen.

Angeblich sind in Hanoi ein paar Mönche seit Urzeiten mit dem Fall $n = 64$ befasst.

Lösung

Für $n = 1$ kein Problem.

Falls man schon weiß, wie es für $n - 1$ geht, dann schafft man mit diesem Rezept die obersten $n - 1$ Scheiben auf den zweiten Stab (die unterste Scheibe fasst man dabei als “Boden” auf.).

Dann legt man die größte nunmehr freie Scheibe auf den dritten Stapel und verschafft unter abermaliger Verwendung der Vorschrift für $n - 1$ die restlichen Scheiben vom mittleren auf den dritten Stapel.

Lösung in Pseudocode

Turm $= \{1, 2, 3\}$

Befehle $= \{(i, j) \mid i, j \in$ Turm$, i \neq j\}$

Befehlsfolge $= \{\vec{b} \mid$ es gibt n so dass $\vec{B} \in$ Befehle$^n\}$

Lösung : $\mathbb{N} \times$ Turm $\times$ Turm $\to$ Befehlsfolge

Lösung $= \mathbf{function}(n, i, j)$

Liefert Befehlsfolge zum Transfer von n Scheiben von i nach j

if $i = j$ **then** leere Befehlsfolge

else let $k = 6 - i - j$ **in**

Lösung$(n-1, i, k)\hat{}(i, j)\hat{}$Lösung$(n-1, k, j)$

Technik der Einbettung

Im Beispiel mussten wir eine allgemeinere Funktion rekursiv definieren.

Versuchen wir, nur die Funktion “verschiebe von 1 nach 2” zu definieren, dann ergibt sich keine rekursive Lösung.

Häufig muss man vor einer rekursiven Lösung das Problem generalisieren.

Diese Technik bezeichnet man als *Einbettung*

Beispiele von Einbettung: Primzahlen

Man soll bestimmen, ob n eine Primzahl ist.

Gesucht istPrim : **nat** $\to$ **bool** mit istPrim$(n) =$ **true** gdw., n prim.

istPrim $=$ **function**(n)**if** $n = 0 \vee n = 1$ **then false else if** $n = 2$ **then true else** ?

Primzahlen

keineTeiler $=$ **function**(n:**nat**, k:**nat**)**bool**

result Stellt fest, ob n keine Teiler im Bereich $k \ldots n-1$ hat

if $k \geq n-1$ **then true**

else $\neg(k \mid n) \wedge$ keineTeiler$(n, k+1)$

istPrim $=$ **function**(n:**nat**)**bool**

$n > 1 \wedge$ keineTeiler$(n, 2)$

Beispiel: Binäre Suche

Man soll ein Wort im Lexikon suchen.

suche : Lexikon $\times$ Wort $\longrightarrow$ **bool**

suche$(l, w) =$ "w kommt in l vor"

Noch nicht detailliert genug.

Wir nehmen an, es gibt eine Funktion

ntesWort : Lexikon $\longrightarrow$ Wort

die das n-te Wort im Lexikon liefert.

Lösung eins: alle durchprobieren

$$\text{sucheBis} = \mathbf{function}(l, w, n{:}\mathbf{nat})\mathbf{bool}$$

$$\quad \mathbf{if}\ n = 0\ \mathbf{then\ false\ else}$$

$$\quad\quad \text{sucheBis}(l, w, n-1) \vee w = \text{ntesWort}(l, n)$$

$$\text{suche} = \mathbf{function}(l, w)$$

$$\quad \text{sucheBis}(l, w, \text{anzahlWörter}(l))$$

Lösung zwei: binäre Suche

```
sucheVonBis = function(l, w, i:int, j:int)bool
    if i > j then false else
    if i = j then ntesWort(l, i) = w else
        let m = ⌊(i + j)/2⌋ in
        let w_m = ntesWort(l, m) in
        if w_m = w then true else
            if w kommt vor ntesWort(l, m) then
                sucheVonBis(l, w, i, m − 1)
            else sucheVonBis(l, w, m + 1, j)
suche = function(l, w)
    sucheVonBis(l, w, 1, anzahlWörter(l))
```

Abstiegsfunktion

Um festzustellen, ob eine rekursiv definierte Funktion für ein Argument definiert ist, kann man eine Abstiegsfunktion verwenden.

Sei

$$f : A \to B$$
$$f = \mathbf{function}(x)\Phi(f, x)$$

eine rekursive Definition einer Funktion $f : A \to B$.

$\Phi(f, x)$ bezeichnet hier den definierenden Term, der sowohl f, als auch x enthält.

Sei $A' \subseteq A$ eine Teilmenge von A und werde in $\Phi(f, x)$ die Funktion f nur für Argumente $y \in A'$ aufgerufen.

Sei $\Phi(f, x)$ immer definiert, wenn $x \in A'$ und $f(y)$ definiert ist für alle Aufrufe $f(y)$ in $\Phi(f, x)$.

Dann muss noch nicht unbedingt gelten $A' \subseteq D(f)$.

Abstiegsfunktion

Sei nun zusätzlich $m : A \to \mathbb{N}$ eine Funktion mit $A' \subseteq D(m)$ mit der folgenden Eigenschaft:

Im Term $\Phi(h, x)$ wird h nur für solche $y \in A'$ aufgerufen, für die gilt $m(y) < m(x)$.

Dann ist $A' \subseteq D(f)$.

Man bezeichnet so ein m als *Abstiegsfunktion*.

Beispiel Fakultät

$$\text{fakultät}(n{:}\textbf{nat}) = \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } n \cdot \text{fakultät}(n-1)$$

Hier nehmen wir $A' = \textbf{nat}$ und $m(x) = x$.

Beispiel keineTeiler

keineTeiler $=$ $\mathbf{function}$(n:**nat**, k:**nat**)**bool**

pre Stellt fest, ob n keine Teiler im Bereich $k \ldots n-1$ hat

if $k \geq n-1$ **then true**

else $\neg k \mid n \wedge$ keineTeiler$(n, k+1)$

Hier setzen wir $A' =$ **nat** $\times$ **nat** und $m(n,k) =$ **if** $k \geq n$ **then** 0 **else** $n-k$.

Wohlfundierte Relationen

Definition Sei M eine Menge. Eine Relation $R \subseteq M \times M$ ist wohlfundiert, wenn es keine unendliche Folge $a_1, a_2, a_3, \ldots$ von Elementen in M gibt sodass $a_{i+1} R a_i$ für alle $i \geq 1$.

Ist R eine wohlfundierte Relation auf einer Menge M, so kann man anstelle einer Abstiegsfunktion $m : A \to \mathbb{N}$ auch eine Abstiegsfunktion $m : A \to M$ wählen, derart dass $m(y) R m(x)$ wenn $f(y)$ in $\Phi(f, x)$ aufgerufen wird.

Beispiele

$M = \mathbb{N}$ und $xRy \Leftrightarrow x < y$.

$M = \mathbb{N}$ und $xRy \Leftrightarrow y = x + 1$

$M = \mathbb{N} \times \mathbb{N}$ und $(x_1, x_2)R(y_1, y_2) \Leftrightarrow x_1 < y_1 \vee x_1 = y_1 \wedge x_2 < y_2$.

Mit dieser wohlfundierten Relation kann man die Ackermannfunktion rechtfertigen:

$$\begin{aligned}
&\text{ackermann} = \mathbf{function}(x{:}\mathbf{nat}, y{:}\mathbf{nat})\mathbf{nat}\\
&\quad \mathbf{if}\ x = 0\ \mathbf{then}\ y + 1\ \mathbf{else}\\
&\qquad \mathbf{if}\ y = 0\ \mathbf{then}\ \text{ackermann}(x - 1, 1)\\
&\qquad\quad \mathbf{else}\ \text{ackermann}(x - 1, \text{ackermann}(x, y - 1))
\end{aligned}$$

Hier wählen wir die Abstiegsfunktion $m(x, y) = (x, y)$.

Verschränkte Rekursion

Manchmal rufen sich zwei Funktionen gegenseitig rekursiv auf. Das ist *verschränkte Rekursion*.

Beispiel

$$\text{gerade} = \mathbf{function}(x{:}\mathbf{nat})\mathbf{if}\ x = 0\ \mathbf{then\ true\ else}\ \text{ungerade}(x-1)$$

$$\text{ungerade} = \mathbf{function}(x{:}\mathbf{nat})\mathbf{if}\ x = 0\ \mathbf{then\ false\ else}\ \text{gerade}(x-1)$$

Es ist $\text{gerade}(x) = (x \bmod 2{=}0)$.

Verschränkte Rekursion über kartesische Produkte

Man kann verschränkte Rekursion durch Produkte simulieren:

$$\begin{array}{l}\text{gerade/ungerade} = \mathbf{function}(x{:}\mathbf{nat})\mathbf{bool} \times \mathbf{bool} \\ \quad \mathbf{if}\ x{=}0\ \mathbf{then}\ (\mathbf{true}, \mathbf{false})\ \mathbf{else} \\ \quad\quad \mathbf{let}\ (g,u) = \text{gerade/ungerade}(x-1)\ \mathbf{in} \\ \quad\quad (u,g)\end{array}$$

Stimmen die Quellen der beiden verschränkt rekursiven Funktionen nicht überein, dann muss man das Produkt der beiden Quellen nehmen.

Diese Simulation legt auch nahe, was von einer Abstiegsfunktion für verschränkt rekursive Funktionen zu fordern ist.

Induktionsbeweise

So wie man eine Funktion rekursiv definieren kann, also durch “Rückgriff” auf andere (hoffentlich schon bekannte) Funktionswerte, so kann man eine Behauptung dadurch beweisen, dass man sie für andere Fälle als bereits bewiesen voraussetzt (*rekursiver Beweis*).

Natürlich muss man dann argumentieren, dass die Kette der rekursiven Rückgriffe irgendwann abbricht, wozu sich wiederum die Abstiegsfunktion anbietet.

Ein solcher rekursiver Beweis mit Abstiegsfunktion ist ein *Induktionsbeweis*.

Induktionsprinzip

Sei

- R eine wohlfundierte Relation auf einer Menge M,
- $m : A \to M$ eine Funktion,
- $P \subseteq A$ eine Teilmenge von A.

Falls für alle $a \in A$ gilt

"a ist in P unter der Annahme, dass alle $y \in A$ mit $m(y)Rm(a)$ in P sind"

dann ist $P = A$.

Beweis des Induktionsprinzips

Äquivalente Formulierung der Bedingung:

"Falls $a \notin P$ dann existiert $y \in A$ mit $m(y)Rm(x)$ und $y \notin P$".

Ein einziges Gegenbeispiel $a \notin P$ zieht also eine unendlich lange Kette von Gegenbeispielen nach sich im Widerspruch zur Wohlfundiertheit von R.

Induktion

Oftmals ist $A = \mathbb{N}$ und $m(n) = n$ und xRy, falls $y = x + 1$.

Hier muss man $0 \in P$ ohne Voraussetzungen zeigen.

Bei $a = y + 1$ darf man aber $y \in P$ schon voraussetzen.

Beispiel

Behauptung: Jedes nur denkbare Verfahren zum Transfer von n Scheiben braucht mindestens $2^n - 1$ Befehle.

Sei P die Menge derjenigen Zahlen n für die das gilt.

$0 \in P$ ist klar, da $2^0 - 1 = 0$.

Sei jetzt $n > 0$. Irgendwann wurde die größte Scheibe verlegt. Dazu aber müssen $n - 1$ Scheiben weggeschafft worden sein (auf den Hilfsstapel). Nach Annahme kostet das mindestens $2^{n-1} - 1$ Befehle. Danach müssen die $n - 1$ Scheiben auf die größte verschafft werden: wieder $2^{n-1} - 1$ Befehle. Insgesamt also $2 \cdot 2^{n-1} - 2 + 1 = 2^n - 1$ Befehle.

Beispiel

Sei $\phi = \frac{\sqrt{5}+1}{2}$. Beachte: $\phi^2 = \phi + 1$.

Behauptung: Es ist $\text{fib}(n) = a\phi^n + b(-\phi)^{-n}$ wobei $a + b = 1$ und $a\phi - b(1/\phi) = 1$.

Sei P die Menge der n für die das wahr ist. Wir wählen $m(x) = x$ und $yRx \Leftrightarrow y < x$.

Es ist $0 \in P$ und $1 \in P$ (Nach Def. von a, b; bei Zweifel nachrechnen)

Wenn $n \geq 2$, dann
$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2) = a\phi^{n-1} + b(-\phi)^{-n+1} + a\phi^{n-2} +$
$b(-\phi)^{-n+2} = a\phi^n(\phi^{-1} + \phi^{-2}) + b(-\phi)^{-n}(-\phi + \phi^2) = a\phi^n + b(-\phi)^{-n}$.

Also $n \in P$ und $P = \mathbb{N}$.

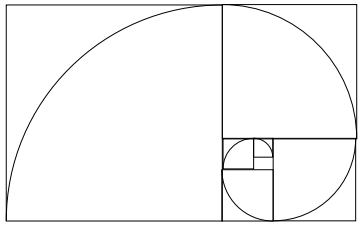
Kettenbruch und Kettenwurzel

Es ist $\phi^2 = 1 + \phi$, also $(1/\phi) = \frac{1}{1+(1/\phi)}$, also

$$1/\phi = \cfrac{1}{1 + \cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{\ddots}}}}}$$

Außerdem $\phi = \sqrt{1+\phi}$, also $\phi = \sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\ldots}}}}$.

Die Zahl $\phi = \frac{\sqrt{5}+1}{2}$ heißt Goldener Schnitt.



Pflasterung

Es gelte, ein $2 \times n$ Rechteck mit Dominos der Größe 2×1 zu pflastern.

Behauptung: Die Anzahl der Möglichkeiten, das zu tun beträgt $\text{fib}(n)$.

Sei P die Menge der n, für die das gilt.

Sei n fest aber beliebig vorgegeben. Wenn $n \leq 1$, dann $n \in P$.

Wenn $n > 1$, dann gibt es zwei Möglichkeiten, die linke obere Ecke[a] zu pflastern. Entweder mit einem senkrechten oder mit einem waagrechten Domino. Im ersten Fall bleibt ein Rechteck der Größe $2 \times (n-1)$ zu pflastern. Da $n - 1 \in P$ gibt es dafür $\text{fib}(n-1)$ Möglichkeiten. Im anderen Fall ist man gezwungen auf die linke untere Ecke auch ein waagrechtes Domino zu legen—es bleibt ein Rechteck der Größe $2 \times (n-2)$, welches man auf $\text{fib}(n-2)$ Arten pflastern kann.

Insgesamt hat man also $\text{fib}(n-1) + \text{fib}(n-2) = \text{fib}(n)$ Möglichkeiten und es ist $n \in P$.

[a] Höhere Mächte befahlen: Linke obere Ecke schwarz malen (SIGMAR POLKE)

Exponentiation

$$\exp = \textbf{function}(x\text{:}\textbf{real}, n\text{:}\textbf{nat})\textbf{real}$$

if $n{=}0$ **then** 1

else n gerade **then** $\exp(x, n \text{ div } 2)^2$

else $\exp(x, n \text{ div } 2)^2 \cdot x$

Hier ist n div 2 die ganzzahlige Division, z.B., $5 \text{ div } 2 = 2$.

Man beweise: $\exp(x, n) = x^n$.

Polymorphe Funktionen

Manchmal gestattet ein Term mehrere Typisierungen. Man kann ihm dann einen Typ mit Typvariablen zuweisen.

Typvariablen bezeichnen wir mit griechischen Buchstaben α, β, γ.

Ein Typ mit Typvariablen heißt *polymorpher Typ* (auch *Polytyp*).

Ein Typ ohne Typvariablen heißt auch *monomorpher Typ* (auch *Monotyp*).

Beispiele:

$$\text{erstes} : \alpha \times \beta \to \alpha$$
$$\text{erstes} = \mathbf{function}(x, y)x$$

$$\text{tausch} : \alpha \times \beta \to \beta \times \alpha$$
$$\text{tausch} = \mathbf{function}(x{:}\alpha, y{:}\beta)(y, x)$$

Polymorphe Funktionen

Man kann eine polymorphe Funktionen $f : \mathit{typ} \to \mathit{typ}'$ mit einem Argument x aufrufen, falls es eine Ersetzung der Typvariablen in *typ* gibt, sodass der Typ von x herauskommt.

Das Ergebnis des Aufrufes $f(x)$ hat den den Typ, der sich durch dieselbe Ersetzung der Typvariablen in typ' ergibt.

$$\text{tausch}(7, 13) = (13, 7) : \textbf{nat} \times \textbf{nat}$$

$$\text{tausch}(\textbf{true}, -13) = (-13, \textbf{true}) : \textbf{int} \times \textbf{bool}$$

$$\text{tausch}(3.14, \textbf{false}) = (3.14, \textbf{false}) : \textbf{real} \times \textbf{bool}$$

Funktionen höherer Ordnung

Wir haben schon gesehen, dass Funktionen als Argumente anderer Funktionen auftreten können.

Funktionen können auch als Wert zurückgegeben werden:

$$\begin{aligned}\text{plus} = \mathbf{function}(x)\\ \quad \mathbf{function}(y)x + y\end{aligned}$$

$\text{plus}(2) = \mathbf{function}(y)y + 2$, also die Funktion “addiere zwei dazu”.

Funktionen höherer Ordnung

Allgemein kann man zu jeder Funktion

$$f_1 : typ_1 \times \cdots \times typ_n \to typ$$

eine Funktion

$$f_2 : typ_1 \to typ_2 \to \cdots \to typ_n \to typ$$

definieren durch

$$f_2 = \mathbf{function}(x_1)\mathbf{function}(x_2)\ldots\mathbf{function}(x_n)f_1(x_, \ldots, x_n)$$

Es ist

$$f_2\ x_1\ x_2\ \ldots x_n = f_1(x_1, \ldots, x_n)$$

Wir bezeichnen f_2 als das *Currying* von f_1 und f_1 als das *Uncurrying* von f_2. (Nach dem Logiker HASKELL CURRY (1900-1982))

Funktionen höherer Ordnung

CURRY (1900–1982)

SCHÖNFINKEL

Schönfinkel gilt als der eigentliche Erfinder des “Currying”.

Iteration und Komposition

$$\text{iteriere} = \mathbf{function}(f{:}\alpha \to \alpha, x{:}\alpha, n{:}\mathbf{nat})\alpha$$
$$\quad \mathbf{if}\ n = 0\ \mathbf{then}\ x\ \mathbf{else}\ \text{iteriere}(f, f(x), n-1)$$

Es ist

$$\text{iteriere}(f, x, n) = \underbrace{f(f(f(\ldots f}_{n\text{ Mal}}(x)\ldots)$$

Andere Notation: $\text{iteriere}(f, x, n) = f^n(x)$.

$$\text{komponiere} = \mathbf{function}(g{:}\beta \to \gamma, f{:}\alpha \to \beta)\alpha \to \gamma$$
$$\quad \mathbf{function}(x{:}\alpha)g(f(x))$$

Andere Notation: $\text{komponiere}(g, f) = g \circ f$.

Noch ein Beispiel zur Induktion

Seien $f : A \to B$ und $g : B \to A$ beliebige Funktionen.

Für alle n ist

$$g(\text{iteriere}(\text{komponiere}(f, g), x, n)) = \text{iteriere}(\text{komponiere}(g, f), g(x), n)$$