Kapitel 2. Konzepte funktionaler Programmierung

Der Funktionsbegriff

A und B seien Mengen. Eine *Funktion* f von A nach B ist eine Zuordnung von *genau einem* Element $y = f(x)$ zu jedem Element x einer Teilmenge A' von A.

A' ist der *Definitionsbereich* von f.

Ist $A' \neq A$, so ist f eine partielle Funktion.

Man schreibt $A' = D(f)$.

Beispiele

$f : \mathbb{R} \to \mathbb{R}$

$f(x) = 1/x$

$D(f) = \mathbb{R} \setminus \{0\}$

$A = B =$ Endliche Folgen von 0en und 1en.

$$f(x) = \begin{cases} 0w, \text{ falls } x = 1w \text{ für ein } w \in A \\ \text{undefiniert sonst} \end{cases}$$

$D(f) = \{1w \mid w \in A\}$

$g : \mathbb{N} \to \mathbb{N}$

$$g(x) = \begin{cases} x/2, \text{ falls } x \text{ gerade} \\ 3x + 1, \text{ sonst} \end{cases}$$

$D(g) = \mathbb{N}$

Beispiele

$$f : \mathbb{N} \to \mathbb{N}$$

$$f(x) = \begin{cases} \text{das kleinste } n \in \mathbb{N} \text{ so dass } \underbrace{g(g(g(\ldots g}_{n \text{ Mal}}(x)\ldots) = 1, \text{ falls es existiert} \\ \\ \text{undefiniert sonst} \end{cases}$$

Es ist ein offenes Problem, ob $D(f) = \mathbb{N}$

Z.B.: $f(27) = 111$.

Terminologie

$f : A \to B, D(f) = A'$.

A heißt *Quelle*, B heißt *Ziel* von f.

Wenn $a \in D(f)$, so ist $f(a)$ der *Wert* der *Anwendung* von f auf das *Argument* a.

Man schreibt statt $f(a)$ manchmal auch

fa	Präfixnotation
af	Postfixnotation

Funktionen mit zwei Argumenten

Sind A_1 und A_2 Mengen, so bildet man das *kartesische Produkt*

$$A_1 \times A_2 = \{(a_1, a_2) \mid a_1 \in A_1 \text{ und } a_2 \in A_2\}$$

Ist $f : A_1 \times A_2 \to B$, so kann man f auf Paare anwenden.

Z.B.:

$$f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$
$$f(x, y) = x + 2y^2$$

Man schreibt *nicht* $f((x, y))$.

Für solche Funktionen gibt es auch die *Infixnotation* $x\,f\,y$ für $f(x, y)$. Etwa, wenn $f = '+'$.

Eine Funktion von $A_1 \times A_2$ nach B heißt *zweistellig*.

Funktionen mit mehreren Argumenten

Sind $A_1, \ldots A_n$ Mengen, so bildet man das kartesische Produkt

$$A_1 \times \cdots \times A_n = \{(a_1, \ldots, a_n) \mid a_i \in A_i \text{ für } i = 1 \ldots n\}$$

Die Elemente von $A_1 \times, \ldots, \times A_n$ heißen n-*Tupel* (Verallg. von Tripel, Quadrupel, Quintupel, Sextupel,...).

Zum Beispiel

$$\text{fahrtzeit} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

Solch eine Funktion heißt n-*stellig* (Vokabeln: *Stelligkeit*, n-*ary*, *arity*)

Nur äußerst selten ist $n > 6$.

Kartesische Produkte als Ziel

Es gibt auch Funktionen, die Paare oder gar n-Tupel zurückliefern.

$$\text{divmod} : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}$$

$$D(\text{divmod}) = \mathbb{N} \times (\mathbb{N} \setminus \{0\})$$

$$\text{divmod}(a, b) = (q, r), \text{ wobei } a = qb + r \text{ und } q, r \in \mathbf{N} \text{ und } r < b$$

Typen

Typen = Mengen gleichartiger Daten, die von Funktionen verarbeitet werden.

Basistypen: $\mathbb{N}, \mathbb{R}, \mathbb{Z}, \mathbb{B} = \{\text{true}, \text{false}\}, \ldots$

Zusammengesetzte Typen: $A_1 \times \cdots \times A_n$

In Pseudocode schreiben wir auch
`nat, real, int, bool, A_1 * ... * A_n`.

Terme

Terme sind Ausdrücke mit Variablen (Platzhaltern).

Beispiel eines Terms: $(s_{\mathrm{an}} - s_{\mathrm{ab}}) \cdot 60 + m_{\mathrm{an}} - m_{\mathrm{ab}}$.

Dieser Term enthält die Variablen $s_{\mathrm{ab}}, m_{\mathrm{ab}}$ und $s_{\mathrm{an}}, m_{\mathrm{an}}$

Kennt man die Werte der Variablen, die in einem Term vorkommen, so kann man den Wert des Terms ausrechnen.

Damit der Term sinnvoll ist, muss man zu jeder Variablen angeben, aus welcher Menge ihre Werte kommen sollen.

Man bezeichnet diese Menge als den *Typ* der Variablen.

Außerdem: *Typ des Terms* = Menge aus der die Werte des Terms zu gegebenen Werten der Variablen entstammen.

Im Beispiel sollen $s_{\mathrm{ab}}, m_{\mathrm{ab}}$ und $s_{\mathrm{an}}, m_{\mathrm{an}}$ jeweils den Typ $\mathbb{N}$ haben. Der Typ des Terms für die Fahrzeit hat auch den Typ $\mathbb{N}$.

Beispiele

Sei x eine Variable vom Typ $\mathbb{R}$ und p eine Variable vom Typ $\mathbb{Z}$.

Was sind die Typen der Terme $x + x$, 1, $\lceil x \rceil$, x^p?

Beachte: für bestimmte erte kann ein Term undefiniert sein, etwa x/y für $y = 0$.

Funktionen aus Termen

Seien $x_1, \ldots, x_n$ Variablen vom Typ $A_1, \ldots, A_n$ (jeweils).

Sei t ein Term vom Typ B, der alle oder einige dieser Variablen enthält.

Wir bilden eine Funktion

$\mathbf{function}(x_1, \ldots, x_n)t : A_1 \times \cdots \times A_n \to B$

$(\mathbf{function}(x_1, \ldots, x_n)t)(a_1, \ldots, a_n) =$ Wert des Terms t, falls x_i den Wert a_i hat.

Beispiel:
fahrtzeit $= \mathbf{function}(s_{\text{ab}}, m_{\text{ab}}, s_{\text{an}}, m_{\text{an}})(s_{\text{an}} - s_{\text{ab}}) \cdot 60 + m_{\text{an}} - m_{\text{ab}}$

Man schreibt statt $\mathbf{function}(x_1, \ldots, x_n)t$ auch $\lambda(x_1, \ldots, x_n)t$ (vgl. Logo des TCS Lehrstuhls).

Funktionstypen

Mit $A_1 \times \cdots \times A_n \to B$ bezeichnet man die Menge der Funktionen von $A_1 \times \cdots \times A_n$ nach B.

Solche Mengen können als Typ von Variablen und als Typ von Termen auftreten.

Zum Beispiel ist $\mathbf{function}(x)x$ ein Term vom Typ $A \to A$ falls x eine Variable vom Typ A ist.

Freie Variablen

Der Term $\mathbf{function}(x)x$ enthält keine Variablen in dem Sinne, dass sein Wert von ihnen abhängt.

Die Variable x ist nämlich in $\mathbf{function}(x)x$ *gebunden*.

Ebenso ist x in $\int_0^1 e^x \mathrm{d}x$ gebunden.

Variablen, die “echt” in einem Term vorkommen, bezeichnet man als *freie* Variablen.

Freie Variablen in Funktionstermen

Ein Funktionsterm kann trotzdem freie Variablen enthalten:

$$\mathbf{function}(x)x + y$$

enthält die Variable y. Der Wert des Terms hängt vom Wert der Variablen y ab (und ist dann die Funktion “addiere y”).

Anderes Beispiel: der Term $\int_0^\infty e^{-st} f(t)\mathrm{d}t$ enthält die Variable s vom Typ $\mathbb{R}$ und die Variable f vom Typ $\mathbb{R} \to \mathbb{R}$.

Lokale Definitionen

Terme dürfen auch informelle deutsche Beschreibungen enthalten:

das kleinste gemeinsame Vielfache von x und y

ist ein Term mit den Variablen x und y.

$x + y$, wobei $y = 10$

ist ein Term mit der Variablen x. Die Variable y ist wiederum *gebunden*.

0, falls $x \leq 0$; 1, falls $x > 0$

ist ein Term mit der freien Variablen x.

Standardnotation

In Pseudocode (und später in der Programmierung) verwendet man gern standardisierte Notation für solche Terme.

Statt

t_1, wobei $x = t_2$

schreiben wir

let $x = t_2$ **in** t_1

Statt

t_1, falls Bedingung A und t_2 sonst

schreiben wir

if Bedingung A **then** t_1 **else** t_2

Beispiel

Erweiterte Fahrzeitberechnung:

$$\begin{array}{l}\textbf{function}(s_{\mathrm{ab}}, m_{\mathrm{ab}}, s_{\mathrm{an}}, m_{\mathrm{an}}) \\ \quad \textbf{let } \mathit{zwErg} = (s_{\mathrm{an}} - s_{\mathrm{ab}}) \cdot 60 + m_{\mathrm{an}} - m_{\mathrm{ab}} \textbf{ in} \\ \quad \textbf{let } \mathit{minProTag} = 24 \cdot 60 \textbf{ in} \\ \quad \textbf{if } \mathit{zwErg} \geq 0 \textbf{ then } \mathit{zwErg} \textbf{ else } \mathit{zwErg} + \mathit{minProTag}\end{array}$$

Beachte: im Skript werden **let** Konstrukte mit **end** abgeschlossen. Wir machen das nicht, es ist aber auch kein Fehler.

Formulierung von Bedingungen

Eine *Bedingung* ist ein Term vom Typ $\mathbb{B} = \{\text{true}, \text{falsch}\}$ (im Pseudocode geschrieben `bool`).

Zur Formulierung von Bedingungen verwenden wir z.B. die Vergleichsoperationen

$$<, >, \leq, \geq \colon \mathbb{N} \times \mathbb{N} \to \mathbb{B}$$

Für diese verwendet man die Infixnotation, also $3 < x$ statt $< (3, x)$.

Bedingungen kann man mit den logischen Operationen $\wedge, \vee, \neg$ zusammensetzen.

$x < 7 \wedge x > 8$.

Typannotate

Man kann Quelle und Ziel eines Funktionsterms auch in diesen aufnehmen:

Statt

$$f : A_1 \times \cdots \times A_n \to B$$
$$f = \mathbf{function}(x_1, \ldots, x_n)t$$

schreiben wir auch

$$f = \mathbf{function}(x_1{:}A_1, \ldots, x_n{:}A_n)B\ t$$

oder

$$f = \mathbf{function}(x_1{:}A_1, \ldots, x_n{:}A_n)B{:}t$$

Notation aus dem Skript

Beispiele

$$\textbf{function}(s_{\mathrm{ab}:\textbf{nat}}, m_{\mathrm{ab}}:\textbf{nat}, s_{\mathrm{an}}:\textbf{nat}, m_{\mathrm{an}}:\textbf{nat})\textbf{nat}$$

let *zwErg* $= (s_{\mathrm{an}} - s_{\mathrm{ab}}) \cdot 60 + m_{\mathrm{an}} - m_{\mathrm{ab}}$ **in**

let *minProTag* $= 24 \cdot 60$ **in**

if *zwErg* ≥ 0 **then** *zwErg* **else** *zwErg* $+$ *minProTag*