Übersicht

4.1 Ausdrücke

4.2 Funktionale Algorithmen

4.3 Anweisungen

4.4 Imperative Algorithmen

4.2 Funktionale Algorithmen

- Das Konzept der Ausdrücke erlaub uns nun, einfache funktionale Algorithmen zu entwerfen.
- Zur Veranschaulichung dient folgendes Beispiel aus der elementaren Physik:

  Ein frei beweglicher Körper mit der Masse *m* (>0) werde aus der Ruhelage ein Zeit *t* lang mit einer auf ihn einwirkenden konstanten Kraft *k* bewegt. Gesucht ist ein Algorithmus, der (in Abhängigkeit von *m*, *t* und *k*) die Strecke *s* bestimmt, um die der Körper aus seiner ursprünglichen Lage fortbewegt wird.
  - Die Datendarstellung ist in diesem Beispiel sehr einfach: Die Größen *m*, *t* und *k* sind offensichtlich aus $\mathbb{R}$ (mit $m > 0$ und $t \geq 0$)
  - Der gesuchte Algorithmus kann damit als Abbildung (Funktion)

    $$\textit{Strecke}: \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

    formuliert werden, wobei die Berechnungsvorschrift explizit (als Ausdruck) unter Verwendung von Grundoperationen angegeben werden soll

4.2 Funktionale Algorithmen

- Die Idee für dieses Algorithmus ergibt sich aus folgenden einfachen Gesetzen:
  - Die auf einen Körper mit Masse *m* einwirkende Kraft *k* erzeugt eine konstante Beschleunigung

    $$b = k/m$$

  - Der in der Zeit *t* zurückgelegte Weg bei einer Bewegung mit konstanter Beschleunigung *b* ist

    $$s = ½ * b * t^2$$

  - Einsetzen liefert

    $$Strecke(m,t,k) = (k * t^2) \ / \ (2 * m)$$

  - Dies beschreibt, wie man *s* durch Anwendung verschiedener Grundoperationen auf *m*, *t* und *k* bestimmen kann

4.2 Funktionale Algorithmen

- Der entsprechende Algorithmus ist:

  Algorithmus *Strecke* : $\mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$

  (Bedingung: $m > 0,\ t \geq 0$)

  Strecke(m,t,k) = (*k*t*t*) / (2*m*)

  dabei ist

  - *Strecke* der Name des Algorithmus (der Funktion)
  - *m*, *t* und *k* Variablen der Sorte $\mathbb{R}$, die sog. (*formalen*) *Parameter* von *Strecke.*
  - *(k*t*t) / (2*m)*“ ein Ausdruck (mit implizitem Typcast) der Sorte $\mathbb{R}$ auch (Funktions-) *Rumpf* genannt (die Berechnungsvorschrift); hier kann ein beliebiger Ausdruck stehen; als Variablen für diesen Ausdruck sind zunächst nur die formalen Parameter des Algorithmus erlaubt (**Warum?**)
  - Einschränkungen des Definitionsbereichs werden *Vorbedingungen* genannt

4.2 Funktionale Algorithmen

- Den Algorithmus bzw. die Funktion *Strecke* kann nun auch innerhalb anderer Algorithmen verwendet werden genauso wie Basisoperationen
- Dazu kann die Syntax von Ausdrücken entsprechend erweitert werden
  - Sind $a_1, \ldots, a_n$ Ausdrücke der Sorten $S_1, \ldots, S_n$ und f eine Funktion (Algorithmus) mit der Signatur $S_1 \times \ldots \times S_n \rightarrow S_0$ (wobei $S_0, \ldots, S_n \in S$), dann ist $f(a_1, \ldots, a_n)$ ein Ausdruck der Sorte S_0.
- Beispiel:
  - Die Arbeit berechnet sich aus Kraft multipliziert mit der Strecke

$$\text{Algorithmus } \mathit{Arbeit} : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

(Bedingung: $m > 0,\ t \geq 0$)

$$\mathit{Arbeit}(m,t,k) = k * \mathit{Strecke}(m,t,k)$$

  - Im Rumpf des Algorithmus wird ein Ausdruck mit der Funktion *Strecke* gebildet. Die Funktion/der Algorithmus *Strecke* wird hier *aufgerufen*

4.2 Funktionale Algorithmen

- Auch die Ausführung eines Algorithmus ist ein Funktionsaufruf. Was passiert bei so einem Funktionsaufruf?
- Der Aufruf des Algorithmus *Strecke* mit konkreten Werten ist nichts anderes als eine Substitution:
  - Der Algorithmus wird durch einen Ausdruck definiert.
  - Der *Wert* dieses Audrucks ergibt sich aus der Substitution der Eingabevariablen durch die entsprechenden Werte, mit denen der Algorithmus aufgerufen wird
  - Diese Substitution σ wird auch *Variablenbelegung* genannt; die Eingabevariablen werden jeweils mit einem Literal der entsprechenden Sorte belegt
  - $V(\sigma)$ bezeichnet die Menge der (Eingabe-)Variablen in dieser Substitution σ
  - z.B. beim Aufruf von *Strecke*(1.0,2.0,3.0) werden die Variablen m,t,k mit den Literalen $1.0,2.0,3.0 \in \mathbb{R}$ belegt, d.h. $V(\sigma) = \{m,t,k\}$

    Wir schreiben $\sigma = [m/1.0,\ t/2.0,\ k/3.0]$

4.2 Funktionale Algorithmen

Gegeben eine Substitution/Variablenbelegung σ

($V(\sigma)$ bezeichnet die Variablen in σ)

Rekursive Definition des *Wertes* $W_\sigma(u)$ *eines Ausdrucks* u bzgl. σ:

- Wenn u eine Variable $v \in V(\sigma)$ ist, so ist $W_\sigma(u) = u\sigma$
- Wenn u ein Literal op ist, so ist $W_\sigma(u) = op$
- Wenn u ein Anwendung eines n-stelligen Operators $op(a_1, \ldots, a_n)$ ist, so ist $W_\sigma(u) = \mathrm{op}(W_\sigma(a_1), \ldots, W_\sigma(a_n))$
- Wenn u ein Funktionsaufruf $f(a_1, \ldots, a_n)$ mit Funktionsrumpf r ist, so ist $W_\sigma(u) = W_\sigma(r)$

- Bemerkungen
  - $W_\sigma(u)$ ergibt als Wert ein Objekt der Sorte von u, bezeichnet durch das entsprechende Literal der Sorte
  - Wir schreiben $W(u)$ statt $W_\sigma(u)$, wenn σ aus dem Kontext klar ersichtlich ist
  - Wenn u eine Variable $v \notin V(\sigma)$ ist, so ist $W_\sigma(u)$ *undefiniert*

4.2 Funktionale Algorithmen

- Beispiel: Algorithmus $f: \mathbb{N}_0 \rightarrow \mathbb{N}_0$

  $f(x) = (x * x) + 4$

  Der Aufruf $f(2)$ entspricht einer Substitution $\sigma=[x/2]$ mit $V(\sigma) = \{x\}$

  Der Wert $W((x * x) + 4)$

  $= W(x * x) + W(4)$

  $= (W(x) * W(x)) + 4$

  $= (x\ [x/2] * x\ [x/2]) + 4$

  $= (2 * 2) + 4$

  $= 4 + 4$

  $= 8$

4.2 Funktionale Algorithmen

- Aufruf *Strecke*(1.0,2.0,3.0) entspricht der Substitution σ=[*m*/1.0, *t*/2.0, *k*/3.0]

$$
\begin{aligned}
W((k*t*t) / (2*m)) &= W(k*t*t) / W(2*m) \\
&= (W(k) * W(t) * W(t)) / (W(2) * W(m)) \\
&= (k\ [k/3.0] * t\ [t/2.0] * t\ [t/2.0]) / (2*m\ [m/1.0]) \\
&= (3.0*2.0*2.0) / (2*1.0) \\
&= 12.0 / 2.0 \\
&= 6.0
\end{aligned}
$$

- Aufruf *Arbeit*(1.0,2.0,3.0) entspricht der Substitution σ=[*m*/1.0, *t*/2.0, *k*/3.0]

$$
\begin{aligned}
W(k * Strecke(m,t,k)) &= W(k) * W(Strecke(m,t,k)) \\
&= k\ [k/3.0] * W((k*t*t) / (2*m)) \\
&= 3.0 * \ldots \\
&= 3.0 * 6.0 \\
&= 18.0
\end{aligned}
$$

4.2 Funktionale Algorithmen

- Um Fallunterscheidung zu modellieren, benötigen wir nun noch das Konzept der *bedingten Ausdrücke* (*Terme*)
- Dazu erweitern wir die induktive Definition der Ausdrücke (Folie 13) um folgenden Fall
  - Voraussetzung: die Menge der Sorten S enthält die Sorte $\mathbb{B}$
  - Ist b ein Ausdruck der Sorte $\mathbb{B}$ und sind a_1 und a_2 Ausdrücke der Sorten $S_0 \in S$, dann ist

    if b **then** a_1 **else** a_2 **endif**

    ein Ausdruck der Sorte S_0.

    b heißt *Wächter*, a_1 und a_2 heißen *Zweige*.
- Bemerkung: der Ausdruck a_2 kann offenbar wiederum ein bedingter Ausdruck (der Sorte S_0) sein, d.h. man kann bedingte Ausdrücke (beliebig) ineinander schachteln

4.2 Funktionale Algorithmen

- Beispiele
  - Für $x \in \mathbb{Z}$ ist der Absolutbetrag von x bestimmt durch

    Algorithmus *ABS*: $\mathbb{Z} \to \mathbb{N}_0$

    $ABS(x) =$ **if** $x \geq 0$ **then** x **else** $-x$ **endif**

    (In beiden Zweigen des bedingten Ausdrucks finden implizite Typcasts statt)
  - Die der Größe nach mittlere von drei Zahlen x,y,z ist bestimmt durch

    Algorithmus *Mitte*: $\mathbb{N}_0 \to \mathbb{N}_0$

    $Mitte(x,y,z) =$ **if** $x<y \wedge y<z$ **then** y **else**
    if $x<z \wedge z<y$ **then** z **else**
    if $y<x \wedge x<z$ **then** x **else**
    if $y<z \wedge z<x$ **then** z **else**
    if $z<x \wedge x<y$ **then** x **else** y **endif**
    endif endif endif endif

4.2 Funktionale Algorithmen

- Der Wert $W(u)$ eines bedingten Ausdrucks u

  if b **then** a_1 **else** a_2 **endif**

  ist abhängig von W(b):
  - Ist $W(b)$ = TRUE, so ist $W(u) = W(a_1)$
  - Ist $W(b)$ = FALSE, so ist $W(u) = W(a_2)$
- Beispiel

  Aufruf von *ABS*(-3) entspricht einer Substitution $\sigma=[x/-3]$

  W(**if** $x \geq 0$ **then** x **else** $-x$ **endif**) = W(-x) (else-Zweig)

  da W($x \geq 0$) $= (x \geq 0)[x/-3]$

  $= x\,[x/-3] \geq 0\,[x/-3]$

  $= -3 \geq 0 =$ FALSE

  W($-x$) $= -x[x/-3]$

  $= --3 = 3$

4.2 Funktionale Algorithmen

- Rekursion
  - Mit bedingten Ausdrücken lassen sich nun auch rekursive Funktionen (Algorithmen) formulieren.
  - Beispiel: Algorithmus *FAK*: $\mathbb{N}_0 \rightarrow \mathbb{N}_0$

    FAK(*n*) = **if** *n*=0 **then** 1 **else** *n***FAK*(*n*-1) **endif**

    berechnet die Fakultät einer natürlichen Zahl
  - Der Wert der Funktion für einen konkreten Aufruf ist mit den bisherigen Konzepten berechenbar.
  - Damit ist obige Funktion auch wirklich ein Algorithmus zur Berechnung der Fakultätsfunktion
  - Der Aufruf von FAK(3) entspricht wiederum einer Variablenbelegung $\sigma[n/3]$; die Auswertung ist schematisch auf der nächsten Folie dargestellt (statt $\sigma[n/k]$ steht „für *n*=*k*“)

4.2 Funktionale Algorithmen

1) $W(FAK(3))$
   $= W(\text{if } n=0 \text{ then } 1 \text{ else } n*FAK(n-1) \text{ endif})$ für $n=3$
   $= W(n*FAK(n-1))$ für $n=3$ (da $W(n=0)=FALSE$ für $n=3$)
   $= W(n)\cdot W(FAK(n-1))$ für $n=3$
   $= 3\cdot W(FAK(2))$.
2) $W(FAK(2))$
   $= W(\text{if } n=0 \text{ then } 1 \text{ else } n*FAK(n-1) \text{ endif})$ für $n=2$
   $= W(n*FAK(n-1))$ für $n=2$ (da $W(n=0)=FALSE$ für $n=2$)
   $= W(n)\cdot W(FAK(n-1))$ für $n=2$
   $= 2\cdot W(FAK(1))$.
3) $W(FAK(1))$
   $= W(\text{if } n=0 \text{ then } 1 \text{ else } n*FAK(n-1) \text{ endif})$ für $n=1$
   $= W(n*FAK(n-1))$ für $n=1$ (da $W(n=0)=FALSE$ für $n=1$)
   $= W(n)\cdot W(FAK(n-1))$ für $n=1$
   $= 1\cdot W(FAK(0))$.
4) $W(FAK(0))$
   $= W(\text{if } n=0 \text{ then } 1 \text{ else } n*FAK(n-1) \text{ endif})$ für $n=0$
   $= W(1)$ (da $W(n=0)=TRUE$ für $n=0$)
   $= 1$.
5) Einsetzen des Ergebnisses von 4) in 3) ergibt: $W(FAK(1))=1\cdot 1=1$.
6) Einsetzen des Ergebnisses von 5) in 2) ergibt: $W(FAK(2))=2\cdot 1=2$.
7) Einsetzen des Ergebnisses von 6) in 1) ergibt: $W(FAK(3))=3\cdot 2=6$.

4.2 Funktionale Algorithmen

- Weitere Beispiele für rekursive Algorithmen
  - Die Berechnung der Summe 0+1+...+n

    Algorithmus *Summe*: $\mathbb{N}_0 \to \mathbb{N}_0$

    Summe(*n*) = **if** *n*=0 **then** 0 **else** *n* + *Summe*(*n*-1) **endif**

  - Die Berechnung der *n*-ten Fibonacci-Zahl

    Algorithmus *FIB*: $\mathbb{N}_0 \to \mathbb{N}_0$

    FIB(*n*) = **if** *n*=0 $\vee$ *n*=1 **then** 1 **else** *FIB*(*n*-1) + *FIB*(*n*-2) **endif**