Werte

Markus Roggenbach

15. November 2002

Inhalt

Vorbemerkungen

Datentypen in Programmiersprachen

Ausdrücke

Vorbemerkungen

Grundbegriffe

- Wert
- (Daten) Typ
- Ausdruck

Ein *Typ* ist
eine Menge von *Werten*,
die in Form von Ausdrücken *manipuliert* werden können.

Warum Datentypen in Programmiersprachen?

gegeben: Berechnunsproblem $P \subseteq E \times A$.
gesucht: Programm für $f : E \rightarrow ?A$ mit

- $\{(x, f(x)) \mid x \in E\} \subseteq P$
- $\forall e \in E : (\exists a \in A : (e, a) \in P) \Rightarrow def(f)$.

Wünsche an P'sprachen:

- große, aber dennoch übersichtliche Vielzahl von DT'en.
- Datentypen müssen „effizient“ implementierbar sein

funktional — reaktiv

Beschreibungsebene: Sicht auf ein System

funktional:

System realisiert eine Funktion auf Datentypen,
meist terminierendes System

Bsp: `compile hugo.p`

reaktiv:

System realisiert ein Verhaltensmuster von Ereignissen
meist nicht terminierende Systeme

Bsp: `Automat = Münze -> Kaffee -> Automat`

Datentypen in Programmiersprachen

Übersicht

I. Einfache Typen

II. Strukturierte Typen:
   (a) Direkt konstruierte Typen
   (b) Rekursiv beschriebene Typen

Einfache Typen

Ein einfacher Typ enthält *atomare* Werte,
die nicht mehr in einfachere Werte unterteilt werden können.

- Wahrheitswerte (Boolean=$\{TT, FF\}$)
- Typen zur Repräsentation von
  natürlichen, ganzen, rationalen, reellen, komplexen Zahlen
- Zeichen (Char)
- String
- Aufzählungstypen

Aufzählungstyp in Pascal

```
type Monat = (Januar, Februar, März,
              April, Mai, Juni,
              Juli, August, September,
              Oktober, November, Dezember)
```

Gängige Normen

Floats: IEEE 754 und IEEE 854

Char:

- ASCII
- Iso-Latin-1

- Unicode
  (‘covers the principal written languages and symbol systems of the world.’)
  `http://www.unicode.org/`

```
0020    SPACE
        * sometimes considered a control code
        * other space characters: 2000-200A
        x (no-break space - 00A0)
        x (zero width space - 200B)
        x (ideographic space - 3000)
        x (zero width no-break space - FEFF)
```

Standardfunktionen: Boolean

typisch: $\wedge, \vee, \neg, \ldots$

wichtig: funktional vollständig

Beachte: Meist keine strikten Funktionen, wie z.B.

$$x \wedge y = \begin{cases} x & ; x = TT \\ y & ; else \end{cases}$$

Direkt konstruierte Typen

Konstruktionen:

- Produkt ($A_1 \times \ldots \times A_n$)
- Summe ($A_1 + \ldots + A_n$)
- Funktionsraum ($A \rightarrow B$)
- Potenzmenge ($\mathcal{P}(A)$) – hier nicht
- Unterbereich ($x \ldots y$) – hier nicht

Produkt-Konstrukt in Pascal: Record

```
type Monat = (Januar, Februar, März, April,
              Mai, Juni, Juli, August, September
              Oktober, November, Dezember)
type Datum = record
               m: Monat;
               t: 1 .. 31
             end
var someday : Datum
begin
  someday.m := Februar;
  someday.t := 31
```

Produkt: Mathematische Ebene

$$\Pi_{i=1}^{n} A_i$$

$$= \quad A_1 \times \ldots \times A_n$$

$$:= \quad \{(a_1, \ldots, a_n) \,|\, a_i \in A_i,\ 1 \leq i \leq n\}$$

$A_1 \ldots A_n$ Mengen.

Notationen:

- $A^0 = Unit := \{void\}$
- $A^n := A \times \ldots \times A$, $n \geq 1$.

Projektion auf eine Produktkomponente:

für $1 \leq k \leq n$:

$$\pi_k : \begin{cases} \Pi_1^n A_i & \rightarrow & A_k \\ (a_1, \ldots, a_n) & \mapsto & \pi_k(a_1, \ldots, a_n) := a_k \end{cases}$$

Summenkonstrukt in SML: datatype

```
datatype number =   exact  of int
                  | approx of real;

val erna = exact  1;
val hugo = approx 1.0;

fun erwin (exact  i) = i
  | erwin (approx r) = round(r);
```

SML-Mini-Demo

Summe: Mathematische Ebene

$$\Sigma_{i=1}^{n} A_i$$

$$= \quad A_1 + \ldots + A_n$$

$$:= \quad \{in_1(a_1) \,|\, a_1 \in A_1\} \cup \ldots \cup \{in_n(a_n) \,|\, a_n \in A_n\}$$

in_i : injektiv

Selektionsfunktionen:

für $1 \leq k \leq n$:

$$select_k : \begin{cases} \Sigma_{i=1}^{n} A_i & \rightarrow & A_k \\ in_j(a_j) & \mapsto & \begin{cases} a_k & ; j = k \\ undef & ; sonst \end{cases} \end{cases}$$

Funktionsraumkonstrukte in Pascal: Feld und Funktionsabstraktion

```
type Vektor = array [1..20] of real


function Fakultaet (n: integer): integer;
if n=0 then Fakultaet:=1
       else Fakultaet:=Fakultaet(n-1) * n
end
```

Funktionsraum: Mathematische Ebene

$$A \to B := \{ f \subseteq A \times B \mid \forall a \in A \exists b \in B : (a, b) \in f, \\ \forall (a, b_1), (a, b_2) \in f : b_1 = b_2 \}$$

A, B Mengen.

Beachte: $A \to B$ enthält nur totale Funktionen.

Fortsetzung von Totalordnungen?

- Produkt
- Summe
- Funkionsraum

Rekursiv beschriebene Typen

Beispiel in SML:

```
datatyp intlist =   nil
                  | cons of int * intlist

datatype inttree =   leaf of int
                   | branch of inttree * inttree
```

Warum rekursiv beschriebene Typen?

mit *endlichen* Ausdruckmitteln
unendliche Typen beschreiben

- Isormorphie
- Rekursive Gleichungen – Syntax
- Rekursive Gleichungen – Semantik

Isomorphe Mengen – Definition

Mengen A, B heißen *isomorph,*

$$A \cong B,$$

wenn es eine bijektive Funktion $f : A \to B$ gibt.

$\cong$ ist eine *Äquivalenzrelation*,
d.h. für alle Mengen A, B, C gilt:

- $A \cong A$. (Reflexivität)
- $A \cong B \Rightarrow B \cong A$. (Symmetrie)
- $A \cong B \wedge B \cong C \Rightarrow A \cong C$. (Transitivität)

Summen, Produkte, Funktionsräume und Isomorphie

Es seien A, B, C Mengen.

$$A \times B \cong B \times A$$
$$(A \times B) \times C \cong A \times (B \times C) \cong A \times B \times C$$

$$A + B \cong B + A$$
$$(A + B) + C \cong A + (B + C) \cong A + B + C$$

$$(A \times B) + (A \times C) \cong A \times (B + C)$$

$$(A \times B) \to C \cong A \to (B \to C)$$
$$(A + B) \to C \cong (A \to C) \times (B \to C)$$

Rekursive Gleichungen – Syntax I

$\mathcal{E}T$: Menge von einfachen Typen.

X : eine Variable für Mengen.

Terme über $\mathcal{E}T$:

kleinste Menge mit

(i) $T \in \mathcal{E}T$ ist ein Term.

(ii) X ist ein Term.

(iii) $t_1 + t_2$, $t_1 \times t_2$ und $t_1 \to t_2$ sind Terme, wenn t_1, t_2 Terme sind

Rekursive Gleichungen – Syntax II

Eine *Rekursive Gleichung* hat die Form

$$X = t,$$

wobei t ein Term ist.

Rekursive Gleichungen – Semantik

Eine Menge M heißt *Lösung* einer rekursiven Gleichung

$$X = t,$$

wenn

$$M \cong t[M/X],$$

wobei $t[M/X]$ aus t durch Ersetzen von X durch M hervorgeht.

Mehrdeutigkeit

Es gilt: $A \times \mathbf{1} \cong A$

Folge: *jede* Menge M ist Lösung der Gleichung

$$X = X \times \mathbf{1}.$$

Die Lösung einer rekursiven Gleichung ist i.A. *nicht* eindeutig!

Wir wählen (in der Regel) die kleinste Lösung unter Mengeninklusion.

Existenz und Gestalt von Lösungen

In einem „geeigneten Rahmen“ gilt:

Es gibt stets eine kleinste Lösung für $X = t$.

Diese ist gegeben durch:

$$\bigcup_{n \geq 0} L_n,$$

wobei

- $L_0 := \{\}$ und
- $L_{i+1} := t[L_i/X],\ i \geq 0.$

Ausdrücke

Ein *Ausdruck* ist ein Programmstück,
das *ausgewertet* wird,
um einen *Wert* zu erhalten.

Klassifikation von Ausdrücken

Literal:
Wert eines *einfachen* Typs
in Pascal: `365 3.14 '?' 'jkl'`

Aggregat: Wert eines *strukturierten* Typs
in SML: `(a*2.0, b/3.5) (11,27,38,127)`

Funktionsaufruf:
Anwenden von

- Standardfunktion,
- Operator,
- Funktionsabstraktion

auf Parameter
in jeder Programmiersprache (?): `a*b + c/d`

bedingter Ausdruck:
IF $<Bedingung>$ THEN $<Ausdruck>$ ELSE $<Ausdruck>$
in SML: `if x > y then x else y`