Programmiersprachen

Werte

Berthold Hoffmann

Studiengang Informatik
Universität Bremen

Sommersemester 2006
(Vorlesung am 8. Mai 2006)

Werte (8. Mai 2006)

1. Datentypen
2. Ausdrücke
3. Typisierung

Werte (8. Mai 2006)

1. Datentypen
   - einfach
   - direkt zusammengesetzt
   - rekursiv zusammengesetzt
2. Ausdrücke
3. Typisierung

Werte / Daten

- je Sprache hat einen universellen Wertebereich $\mathbb{W}$, das ist die Menge aller Werte, mit denen gerechnet werden kann
- $\mathbb{W}$ wird anhand einer Menge $\mathcal{T}$ von Datentypen klassifiziert
- Ein Datentyp (kurz Typ) $t \in \mathcal{T}$ besteht aus
  - einer Wertemenge $\mathbb{W}_t \subseteq \mathbb{W}$
  - endlich vielen Operationen auf $\mathbb{W}_t$
- Es gibt verschiedene Arten von Typen
  - einfach
  - direkt zusammengesetzt
  - rekursiv zusammengesetzt
- Typen können unterschiedlichen Statur haben
  - benutzerdefiniert
  - vordefiniert (in einer Bibliothek)
  - eingebaut (semantisch unverzichtbar, syntaktisch speziell

Einfache Datentypen

- eingabaut
  - Wahrheitswerte
  - ganze Zahlen
- vordefiniert
  - Zeichen
  - Gleitkommazahlen
- benutzerdefiniert
  - Aufzählungen
  - Bereiche

Wahrheitswerte

- Wertemenge $\mathbb{B} = \{\mathit{false}, \mathit{true}\}$
- mit logischen Operationen
  - Negation $\neg\colon \mathbb{B} \to \mathbb{B}$
  - Konjunktion $\wedge\colon \mathbb{B} \times \mathbb{B} \to \mathbb{B}$
  - Disjunktion $\vee\colon \mathbb{B} \times \mathbb{B} \to \mathbb{B}$
  - und weiteren abgeleiteten Operationen
    - Implikation $\to$
    - Äquivalenz $\leftrightarrow$
    - ...
- semantisch eingebaut
  - Auswahl (**if**)
- syntaktisch: wie ein vordefinierter Aufzählungstyp (siehe unten)

Wahrheitswerte: Besonderheiten

- Abkürzende Auswertung (Ada, Haskell)
  $x \wedge y \Leftrightarrow$ **if** x **then** y **else** *false*
  $x \vee y \Leftrightarrow$ **if** x **then** *true* **else** y
- Ungewöhnliche Operationsnamen (Haskell)
  `Bool` ist ein geordneter Typ ("Instanz der Klasse `Eq`")
  - Implikation als "kleiner gleich": $x \to y \Leftrightarrow x \leqslant y$
  - Äquivalenz als "gleich": $x \leftrightarrow y \Leftrightarrow x = y$
- Wahrheitswerte sind ganze Zahlen (C, C++)
  $n = 0 \Leftrightarrow \mathit{false}$, $n \neq 0 \Leftrightarrow \mathit{true}$

Ganze Zahlen

- Wertemenge $\mathbb{Z}$ wird meist nur mit begrenzter Genauigkeit dargestellt: $\mathbb{Z} = \{\mathit{minint}, \ldots, \mathit{maxint}\}$
  - Typischerweise gilt $\mathit{minint} = 2^{31}$ und $\mathit{maxint} = 2^{31} - 1$
- mit Operationen
  - Zählen $\mathit{succ}, \mathit{pred}\colon \mathbb{Z} \to \mathbb{Z}$
  - Arithmetik $+, -, \div,$ mod $: \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$
  - Vergleiche $=, \neq, <, \leqslant, \geqslant, >\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$
- Beliebig genaue ganze Zahlen (z. B. in Haskell) werden wie Zahlen zur Basis 2^{32} dargestellt.
- semantisch eingebaut
  - Auswahl (**case**, **array**)
- syntaktisch: eingebaute Schreibweise für Werte 42

Einzelzeichen

- Wertemenge $\mathbb{C}$ wird entweder mit ASCII oder UNICODE kodiert.
- mit Operationen
  - Zählen *succ*, *pred*: $\mathbb{C} \to \mathbb{C}$
  - Umwandlung von und in Zahlen
    $\mathrm{ord}: \mathbb{C} \to \mathbb{Z}$, $\mathrm{chr}: \mathbb{Z} \to \mathbb{C}$
  - Vergleiche $=, \neq, <, \leqslant, \geqslant, >: \mathbb{C} \times \mathbb{C} \to \mathbb{B}$
- semantisch: einvordefinierter Aufzählungstyp (siehe unten)
  In manchen Sprachen gilt $\mathbb{C} \Leftrightarrow \mathbb{Z}$.
- syntaktisch: eingebaute Schreibweise für Werte '%'

Zeichenketten sind meist zusammengesetzte Typen (siehe unten)

Reelle Zahlen

- Wertemenge $\mathbb{R}$ kann nur lückenhaft und mit begrenzter Genauigkeit dargestellt werden
- mit Operationen
  - Arithmetik $+, -, *, /, **: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$
  - Vergleiche $=, \neq, <, \leqslant, \geqslant, >: \mathbb{R} \times \mathbb{R} \to \mathbb{B}$
  - Logarithmen-, Winkelfunktionen usw.

Aufzählungen

- Eine Aufzählung definiert eine Wertemenge
  $a = \{c_1, \ldots, c_k\}$
- mit Operationen
  - Zählen *succ*, *pred*: $A \to A$
  - Vergleiche $=, \neq, <, \leqslant, \geqslant, >: A \times A \to \mathbb{B}$
- Intern: $\mathbb{W}_a \subseteq \mathbb{Z}$ und $c_i = i$ (oder $c_i = i - 1$), $1 \leqslant i \leqslant k$

Beispiel (ADA)

```
type Color = { Red, Green, Blue }
```


Bereiche

- Ein Bereichstyp $b = t\mathbf{range}u \ldots o$
- definiert eine zusammenhängende Teilmenge eines einfachen Typs
  $WW_b = \{w \in \mathbb{W}_t \mid u \leqslant w \leqslant o\}$
- erbt die Operationen des Grundtyps t
- Bereichsüberschreitung müssen geprüft werden

Beispiel (ADA)

```
subtype SmallFloat = Float range -1.0 .. +1.0;
subtype Nat = Integer range 0 .. Integer'Last;
subtype Pos = Nat range 1 .. Integer'Last;
subtype LC_Letter = Char range 'a' .. 'z';
subtype Cool = Color range Green .. Blue;
```


einfache Datentypen: Zusammenfassung

- Viele einfache Datentypen sind vordefiniert
- Einige sind eingebaut: Sie werden für die Definition bestimmter Konzepte unbedingt gebraucht
  - $\mathbb{B}$ für bedingte Ausdrücke und Befehle
  - $\mathbb{Z}$ für Feldindices und numerische Auswahl
- Die Wertemenge ist meistens eingeschränkt
- Die Kardinalität ($\#$) solcher Typen ist dann endlich
- Zahlentypen können verschieden große Teilmengen haben
  JAVA: **float** $\subseteq$ **double**
- Darstellung nach internationalen Standards
  (IEEE für Zahlen, UNICODE für Zeichen)
- einfache Datentypen sind Abstraktionen der Datentypen, die die Plattform unterstützt

direkt zusammengesetzte Datentypen

- Produkt
  konkret
  - Verbund (**record**, **struct**)
  - Tupel
- Summe
- Funktionsraum
  konkret
  - Felder (**array**)
  - Funktionsprozeduren (**function**)
- Potenzmenge

kartesisches Produkt

- Definition: $p = s \times t$
- Wertemenge $\mathbb{W}_p = \{(x, y) \mid x \in \mathbb{W}_s, y \in \mathbb{W}_t\}$
- Kardinalität $\#(p) = \#(p) \times \#(t)$
- Operationen
  - Selektion $\mathrm{sel}_1^p: \mathbb{W}_p \to \mathbb{W}_s$, $\mathrm{sel}_2^p: \mathbb{W}_p \to \mathbb{W}_t$
- n-Tupel $p = t_1 \times \cdots \times t_n = \prod_1^n t_i$ $(k \geqslant 2)$
- homogene n-Tupel $t^n = \prod_1^n t$ $(n \geqslant 2)$
- Sonderfall Nulltupel $t^0 = \{()\}$ mit $\#(t^0) = 1$
  Nützlicher als man denkt: **Unit** (JAVA: `void`)

Summe (disjunkte Vereinigung)

- Definition: $v = s + t$
- Wertemenmenge
  $\mathbb{W}_v = \{(0, x) \mid x \in \mathbb{W}_s\} \cup \{(1, x) \mid x \in \mathbb{W}_t\}$
- Kardinalität $\#(v) = \#(s) + \#(t)$
- Operationen
  - Variantenabfrage $\mathrm{isS}^v: v \to \mathbb{B}$, $\mathrm{isT}^v: v \to \mathbb{B}$
  - Projektion $\mathrm{proj}_1^v: \mathbb{W}_v \to \mathbb{W}_s$, $\mathrm{proj}_2^v: \mathbb{W}_v \to \mathbb{W}_t$

Potenzmenge

- $m = \wp(t)$
- Wertemenmenge $\mathbb{W}_m = \{s \mid s \subseteq \mathbb{W}_t\}$
- Kardinalität $\#(m) = 2^{\#(t)}$
- Operationen
  - Elementabfrage $(\in)\colon \mathbb{W}_t \times \mathbb{W}_m \to \mathbb{B}$,
  - Mengenoperationen $(\cup), (\cap), (\setminus)\colon \mathbb{W}_m \times \mathbb{W}_m \to \mathbb{W}_m$
- In wenigen Sprachen definiert
  - PASCAL: Potenzmengen über einfachen Basistypen b mit $\#(b) \leqslant 2^8 = 256$
    Lassen sich als Bitketten der Länge 2^8 darstellen (8 Worte)
  - Für allgemeinere Basismengen gibt es keine kanonische Darstellung, die alle Operationen effizient implementiert

Relation und Funktion

- Eine Teilmenge $R \subseteq A \times B$ heißt (zweistellige) Relation
- Statt $(x, y) \in A$ schreiben wir oft $A(x, y)$
- Eine zweistellige Relation R ist eine Abbildung wenn gilt:
  - $\forall x \in A, y, y' \in B : (x, y) \in R \wedge (x, y') \in R \Rightarrow y = y'$ (rechtseindeutig)
  - $\forall x \in \mathbb{W}_S \exists y \in \mathbb{W}_T : (x, y) \in M$ (linkstotal)
- Dann schreiben wir $R\colon A \to B$
- Statt $(x, y) \in R$ schreiben wir $y = R\,x$ (Funktionsanwendung)

Funktionsraum

- Definition: $F = s \to t$
- Wertemenmenge $\mathbb{W}_F = \{f \subseteq \mathbb{W}_{s \times t} \mid f : s \to t\}$
- Kardinalität $\#(F) = \#(t)^{\#(s)}$
- Operationen
  - Funktionsanwendung vom Typ $F \times s \to t$ ausgedrückt durch Hintereinanderschreiben (*juxtaposition*): $f\ a$ $(a \in \mathbb{W}_F, a \in \mathbb{W}_s)$
- mehrere Parameter $\Leftrightarrow$ ein Produktparameter $f : t_1 \times \cdots \times t_n \to t_0$
- Auftreten in Programmiersprachen
  - Felder (*array*) tabellieren Funktionen mit endlichem Wertebereich
  - Funktionsprozeduren implementieren Funktionen mit einem Algorithmus

Zusammengesetzte Datentypen

- Andere Namen
  - Datenstrukturen
  - Typkonstruktoren (Typkonstruktions-Funktionen)
- Datenstrukturen können beliebig geschachtelt werden mit Typausdrücken $\mathcal{T}$ über Typnamen X:
  - Jeder Typname ist ein Typausdruck: $x \in X \Rightarrow x \in \mathcal{T}$
  - für alle $s, t \in \mathcal{T}$ gilt $s \times t, s + t, s \to t \in \mathcal{T}$
- Datenstrukturen können benannt werden: $x = t$ $(x \in X, t \in \mathcal{T})$
- Frage: Wie können Listen und Bäume definiert werden? Antwort: rekursiv!

Isomorphie von Mengen

Definition (Isomorphie)

Mengen A und B heißen isomorph, geschrieben $A \cong B$, wenn es eine bijektive Funktion $f\colon A \to B$ gibt.

Fakt ($\cong$ ist eine Äquivalenzrelation)

$$\begin{array}{rcll} A & \cong & A & \textit{reflexiv} \\ A \cong B & \Rightarrow & B \cong A & \textit{kommutativ} \\ A \cong B \wedge B \cong C & \Rightarrow & A \cong C & \textit{transitiv} \end{array}$$

Definition (Isomorphiklassen)

$[A]_\cong = \{B \mid A \cong B\}$. Elemente von $[A]_\cong$ werden nicht unterschieden.

Isomorphie von Produkten und Summen

Definition (Null und Eins)

$\mathbf{0} = \{\}$ ist die (eindeutige) leere Menge.
$\mathbf{1}$ ist die Isomorphieklasse aller einelementigen Mengen.

Fakt (Isomorphie von Produkten und Summen)

$$\begin{array}{rcll} A \times B & \cong & B \times A & \textit{kommutativ} \\ A \times (B \times C) & \cong & (A \times B) \times C & \textit{assoziativ} \\ \mathbf{0} \times A & \cong & & \textit{neutral} \\ \\ A + B & \cong & B + A & \textit{kommutativ} \\ A + (B + C) & \cong & (A + B) + C & \textit{assoziativ} \\ \mathbf{1} + A & \cong & A & \mathbf{1}\ \textit{neutral} \\ \mathbf{0} + A & \cong & \mathbf{0} & \mathbf{0}\ \textit{absorbiert} \end{array}$$

Isomorphie von Funktionsräumen

Fakt (Produkte und Summen)

$$(A \times B) + (A \times C) \cong A \times (B + C) \quad \textit{distributiv}$$

Fakt (Funktionsräume)

$$\begin{array}{rcll} (A \times B) \to C & \cong & A \to B \to C & \textit{Curry} \\ (A + B) \to C & \cong & (A \to C) \times (B \to C) & \end{array}$$

Rekursive Typdefinitionen

- Definition: $x = t$ wobei x in t auftritt
- "typisch": t ist eine Summe von rekursiven und nicht rekursiven Typen
- Die Wertemenge $\mathbb{W}_x$ ist eine Lösung der Gleichung $x = t$
- Typgleichungen haben im Allgemeinen mehrere Lösungen!
  - kleinster Fixpunkt: Menge aller endlichen Werte
- rekursives Bestimmen des kleinsten Fixpunktes:
  $\mathbb{W}_x = \bigcup_{i \geqslant 0} \mathbb{W}_x^i$ $\quad \begin{array}{rcl} \mathbb{W}_x^0 & = & \{\} \\ \mathbb{W}_x^{i+1} & = & \mathbb{W}_{t[x/\mathbb{W}_i]} \end{array}$ wobei $t[x/W]$ durch Ersetzen von x durch W in t entsteht
- Kardinalität rekursiver Typen ist unendlich

Listen

Beispiel (Zahlen-Listen)

$\text{IntList} = \text{Unit} + \text{Int} \times \text{IntList}$
Wertemenge:

$$\begin{aligned} L_0 &= \{\} \\ L_1 &= \text{Unit} + \text{Int} \times L_0 = \{()\} \\ L_2 &= \text{Unit} + \text{Int} \times L_1 = \{(n, ()) \mid n \in \mathbb{W}_{\text{Int}}\} \\ L_3 &= \text{Unit} + \text{Int} \times L_2 = \{(n, (m, ())) \mid n, m \in \mathbb{W}_{\text{Int}}\} \\ &\ldots \end{aligned}$$

Endliche Listen von Zahlen

Listen

Beispiel (Bäume)

$\text{IntTree} = \text{Unit} + \text{IntTree} \times \text{Int} \times \text{IntTree}$
Wertemenge:

$$\begin{aligned} T_0 &= \{\} \\ T_1 &= \text{Unit} + \text{IntTree} \times \{\} \times \text{IntTree} = \{()\} \\ T_2 &= \text{Unit} + T_1 \times \text{Int} \times T_1 = \{(n, ()) \mid n \in \mathbb{W}_{\text{Int}}\} \\ T_3 &= \text{Unit} + \text{Int} \times L_2 = \{(n, (m, ())) \mid n, m \in \mathbb{W}_{\text{Int}}\} \\ &\ldots \end{aligned}$$

Endliche Bäume mit Zahlen-markierten Knoten

Andere rekursive Typen

- Beispiele
  - Zyklische Listen
  - Blatt-verkettete Bäume
- Unendliche Werte mit endlich vielen verschiedenen Komponenten
  rationale Listen bzw. Bäume
- dargestellt mit Zeigern (siehe Kapitel Speicher)
- abstrakt: als Graphen

Zeichenketten

- Ein (meistens) vordefinierter Datentyp
- Zeichenketten können dargestellt werden als
  - "einfacher" Datentyp (`string` in ML)
  - Feld von Einzelzeichen (PASCAL)
  - Liste von Einzelzeichen (HASKELL)

  Welche Vor- und Nachteile haben diese Lösungen?

Werte (8. Mai 2006)

1 Datentypen

2 Ausdrücke
- Literal und Aggregat
- Variable, Funktionsanwendung, bedingter Ausdruck

3 Typisierung

Ausdruck

- Die Menge $\mathcal{A}$ aller Ausdrücke enthält:
  - Literale (Wert-Notationen) einfacher vordefinierter Datentypen
  - Aggregate (Wertkonstruktoren) zusammengesetzter Datentypen
  - Namen von Werten (Konstante, Variable)
  - Anwendungen von Funktionen auf Argumente
- Ausdrücke sind getypt: $\mathcal{A}_t$ ist die Menge aller Ausdrücke vom Typ $t \in \mathcal{T}$

Literal und Aggregat

- Literal: Notation für Werte einfacher Typen
  - Zahlen: $42 \in \mathcal{A}_{\texttt{Int}}$ und $0.31415 + 1 \in \mathcal{A}_{\texttt{Float}}$
  - Zeichen und Zeichenketten: $'\%' \in \mathcal{A}_{\texttt{Char}}$ und $\texttt{"HelloWorld!"} \in \mathcal{A}_{\texttt{String}}$
- Aggregat: Notation für Werte zusammengesetzter Typen (Wertkonstruktor)
  - Wenn $e \in \mathcal{A}_s$ und $e' \in \mathcal{A}_t$, dann gilt $(e, e') \in \mathcal{A}_{s \times t}$
  - Wenn $e \in \mathcal{A}_s$, dann gilt $(0, e') \in \mathcal{A}_{s+t}$ und wenn $e' \in \mathcal{A}_t$, dann gilt $(1, e') \in \mathcal{A}_{s+t}$
  - Wenn $x \in X_s$ und $e \in \mathcal{A}_t$, dann gilt $\lambda x.e \in \mathcal{A}_{s \to t}$ (X_s ist die Menge der Variablen vom Typ s)

Variable, Funktionsanwendung, bedingter Ausdruck

- Sei C der Kontext eines Ausdrucks, mit $C = (C_t : X_t \to \mathbb{W}_t)_{t \in \mathcal{T}}$.
  Dann gilt für jede Variable $x \in X_t$: $x : \mathcal{A}_t$
- Wenn $f \in \mathcal{A}_{s \to t}$ und $a \in \mathcal{A}_s$, dann ist $(f\ a) \in \mathcal{A}_t$
- Infix-Operationen sind eine spezielle Schreibweise für Funktionen
  $e \oplus e' \Leftrightarrow (\oplus)\ e\ e'$
- Wenn $c \in \mathcal{A}_{\texttt{Bool}}$ und $e, e' \in \mathcal{A}_t$, dann ist **if** c **then** e **else** $e' \in \mathcal{A}_t$
- Weshalb ist **if** keine Funktion?
  Antwort: Weil nur das erste Argument immer ausgewertet wird

Werte (8. Mai 2006)

1 Datentypen
2 Ausdrücke
3 Typisierung

vereinfachende Annahmen

- Die Wertemengen der einfachen Datentypen sind disjunkt
  Auch $\mathbb{W}_{\texttt{Int}} \cap \mathbb{W}_{\texttt{Float}} = \{\}$
  (Umwandlung von ganzen in Gleitkommazahlen ist eine Operation)
  - $\Rightarrow$ jeder Wert gehört zu genau einem Typ
- Alle Typen sind explizit benannt
- Monomorphie: jeder Name hat höchstens einen Typ

Typisierung

- statisch
  - Jeder Name x wird ein Typ t zugeordnet (und weitere Eigenschaften).
  - Dann können x nur Werte $v \in \mathbb{W}_t$ zugeordnet werden.
  - Ausdrücke werden vor der Ausführung auf Typfehler überprüft.
- dynamisch
  - Im Programm wird Namen kein Typ t zugeordnet
  - Dann können ihnen nacheinander Werte verschiedenen Typs zugeordnet werden.
  - Ausdrücke werden während der Ausführung auf Typfehler überprüft.
- gar keine Überprüfung ...
  - ... geht nicht, wegen Operationen wie $+$ und $<$

Typisierung in konkreten Sprachen

- Alle Referenzsprachen (außer PROLOG) sind statisch getypt.
- Aber: In jedem hinreichend mächtigen System von Typregeln gibt es Regeln, die nur dynamisch überprüft werden können.

Beispiel (ADA)

$x \in \mathbb{W}_{\texttt{Nat}}$?

Beispiel (JAVA)

Jede Variable vom Typ t kann auch Werte eines Untertyps u enthalten.

Typäquivalenz

- strukturelle Äquivalenz
  Zwei Typen s und t sind strukturell äquivalent wenn $\mathbb{W}_s = \mathbb{W}_t$
- namentliche Äquivalenz
  Zwei Typen s und t sind namentlich äquivalent wenn $s = t$

Beispiel (ADA)

type S **is** ... $\Rightarrow$ namentliche Äquivalenz
subtype T **is** ... $\Rightarrow$ strukturelle Äquivalenz

Beispiel (HASKELL)

data $S = \ldots$ $\Rightarrow$ namentliche Äquivalenz
type $T = \ldots$ $\Rightarrow$ strukturelle Äquivalenz

Werte (8. Mai 2006)

1 Datentypen
2 Ausdrücke
3 Typisierung

Zusammenfassung

- Datentypen sind Wertemengen mit darauf arbeitenden Operationen
- Datentypen sind entweder einfach
  oder zusammengesetzt – direkt oder rekursiv
- Ausdrücke sind geschachtelte Operationsaufrufe

Nächstes Mal: Speicher

- Variablen
  - Zeiger
  - Lebensdauer
- Befehle

Walhalla der Programmiersprachen

C.A.R. ("Tony") Hoare.
Notes on Data Structuring.
In: *Structured Programming*.
London: Academic Press, 1972.
Erste systematische Behandlung von Datenstrukturen