Löungsblatt 4 zu “Programmiersprachen”

Berthold Hoffmann, Studiengang Informatik (hof@informatik.uni-bremen.de)

Besprechung am 26. April 2010

Rekursive Typen und Typäquivalenz

Bäume

Definieren Sie die Wertemenge des Typs

$$\mathsf{Tree} = \mathbf{1} + \mathsf{Tree} \times \mathsf{Int} \times \mathsf{Tree}$$

Geben Sie die Bäume bis Tiefe 2 an ($\mathbb{W}^2_{\mathsf{Tree}}$).

Allgemein ist die Wertemenge so definiert:

$$\begin{array}{rcl}
\mathbb{W}_{\mathsf{Tree}} & = & \bigcup_{i \geq 0} \mathbb{W}^i_{\mathsf{Tree}} \\
\mathbb{W}^0_{\mathsf{Tree}} & = & \mathbf{0} \\
\mathbb{W}^{i+1}_{\mathsf{Tree}} & = & \mathbb{W}^i_{\mathsf{Tree}} \cup \mathbb{W}_{\mathbf{1}+\mathsf{Tree}^i \times \mathsf{Int} \times \mathsf{Tree}^i} \\
& = & \mathbb{W}^i_{\mathsf{Tree}} \cup \{\mathsf{L}()\} \cup \{\mathsf{B}(t,i,t') \mid t,t' \in \mathsf{Tree}^i \times \mathsf{Int} \times \mathsf{Tree}^i\}
\end{array}$$

Für $i < 3$ heißt das:

$$\begin{array}{rcl}
\mathbb{W}^0_{\mathsf{Tree}} & = & \mathbf{0} \\
\mathbb{W}^1_{\mathsf{Tree}} & = & \mathbb{W}^0_{\mathsf{Tree}} \cup \mathbb{W}_{\mathbf{1}+\mathsf{Tree}^0 \times \mathsf{Int} \times \mathsf{Tree}^0} \\
& = & \mathbf{0} \cup \{\mathsf{L}()\} \cup \{\mathsf{B}(t,i,t') \mid t,t' \in \mathbb{W}^0_{\mathsf{Tree}}, i \in \mathbb{W}_{\mathsf{Int}}\} \\
& = & \{\mathsf{L}\} \\
\mathbb{W}^2_{\mathsf{Tree}} & = & \mathbb{W}^1_{\mathsf{Tree}} \cup \mathbb{W}_{\mathbf{1}+\mathsf{Tree}^1 \times \mathsf{Int} \times \mathsf{Tree}^1} \\
& = & \{\mathsf{L}\} \cup \{\mathsf{B}(t,i,t') \mid t,t' \in \mathbb{W}^1_{\mathsf{Tree}}, i \in \mathbb{W}_{\mathsf{Int}}\} \\
& = & \{\mathsf{L}\} \cup \{\mathsf{B}(\mathsf{L},i,\mathsf{L}) \mid i \in \mathsf{Int}\} \\
\mathbb{W}^3_{\mathsf{Tree}} & = & \mathbb{W}^2_{\mathsf{Tree}} \cup \mathbb{W}_{\mathbf{1}+\mathsf{Tree}^2 \times \mathsf{Int} \times \mathsf{Tree}^2} \\
& = & \{\mathsf{L}\} \cup \{\mathsf{B}(t,i,t') \mid t,t' \in \mathbb{W}^2_{\mathsf{Tree}}, i \in \mathbb{W}_{\mathsf{Int}}\} \\
& = & \{\mathsf{L}\} \cup \{\mathsf{B}(\mathsf{L},i,\mathsf{L}) \mid i \in \mathsf{Int}\} \cup \{\mathsf{B}(\mathsf{L},i,\mathsf{B}(\mathsf{L},j,\mathsf{L})) \mid i,j \in \mathsf{Int}\} \cup \\
& & \{\mathsf{B}(\mathsf{B}(\mathsf{L},j,\mathsf{L}),i,\mathsf{L}) \mid i,j \in \mathsf{Int}\} \cup \{\mathsf{B}(\mathsf{B}(\mathsf{L},j,\mathsf{L}),i,\mathsf{B}(\mathsf{L},k,\mathsf{L})),\mid i,j,k \in \mathsf{Int}\}
\end{array}$$

Zeichenketten

Vorteile und Nachteile einer Definition von Zeichenketten als

1. primitiver Typ String.

   ⊕ Es werden nur *genau* die für Zeichenketten notwendigen Operationen definiert.

   ⊕ Zeichenketten-Operationen und Operationen auf Feldern und Listen werden nicht durcheinander gebracht.

⊖ Viele auch für Felder oder Listen sinnvollen Operationen müssen noch einmal definiert werden.

2. Feld von Zeichen, beispielsweise in Ada:

   ⊕ Diese Darstellung ist *effizient*.

   ⊖ Wenn die Felder nicht *flexibel* sind (ihre Länge bei sich nicht bei jeder Operation ändern kann), ist das Arbeiten mit Zeichenketten *umständlich*.

   ⊖ Zeichenketten-Operationen und Operationen auf Feldern können durcheinander gebracht werden.

   ⊕ Diese Operationen müssen aber nur einmal definiert werden.

3. Liste von Zeichen, beispielsweise in Haskell:

   ⊕ Viele Listenoperationen können auch für Zeichenketten benutzt werden.

   ⊖ Zeichenketten-Operationen und Operationen auf Feldern können durcheinander gebracht werden.

   ⊕ Diese Operationen müssen aber nur einmal definiert werden.

Klassengesellschaft

Vergleichen Sie für mindestens zwei der Programmiersprachen Ada, C++, Java, ML und Haskell, für welche Typen es Aggregate bzw. Infixoperationen?

	Ada	C++	Java	ML	Haskell
Aggregate	+	+	(+)	+	+
Infix-Operationen	+	+	·	+	+

- In Java müssen Aggregate als Konstruktorfunktionen selbst definiert werden.
- In Java können Benutzer keine Infix-Operationen definieren.

Welche Typen können in Funktionen als Parameter bzw. Ergebnisse benutzt werden?

	Ada	C++	Java	ML	Haskell
Parameter	(+)	(+)	(+)	+	+
Ergebnisse	−	−	−	+	+

- In allen imperativen und OO-Sprachen werden funktionale Ergebnisse verboten – wegen der viel aufwändigeren Implementierung.
- In Ada sind Funktionsparameter nur umständlich (als *generische Prozeduren*) und in C++ und Java nur als *Prozedurzeiger* vorhanden.

Welche der Sprachen sind Ihrer Meinung nach *typvollständig?*