Übungen zur "Theorie der Programmierung" - SoSe2015

Freitag 12:15-13:45, Martenstr. 3, Raum 02.133-113

Daniel Hausmann

daniel.hausmann@fau.de

Friedrich-Alexander-Universität Erlangen
Department Informatik

Lehrstuhl 8

June 11, 2015

Übung 6 - Binäre Bäume

data BinTree a **where**
$\quad Leaf : () \rightarrow$ **BinTree** a
$\quad Bin \;\; : \;$ **BinTree** $a \rightarrow a \rightarrow$ **BinTree** $a \rightarrow$ **BinTree** a
$mirror \;\; Leaf = Leaf$
$mirror \;\; (Bin \;\; l \;\; a \;\; r) = Bin \; (mirror \;\; r) \;\; a \;\; (mirror \;\; r)$
$inorder \;\; Leaf = Nil$
$inorder \;\; (Bin \;\; l \;\; a \;\; r) = inorder \;\; l \oplus Cons \; a \; (inorder \;\; r)$

Beweisen Sie mittels struktureller Induktion die folgenden Eigenschaften.
Hinweis: Sie sollten jeweils zwei Induktions-Hypothesen aufstellen!
Ausserdem werden Sie vermutlich in vorherigen Übungen bereits bewiesene Eigenschaften benötigen.

1. $\forall t, \;\; mirror \;\; (mirror \;\; t) = t.$
2. $\forall t, \;\; inorder \;\; (mirror \;\; t) = reverse \;\; (inorder \;\; t).$

Sortierte Signatur

Eine **sortierte Signatur** $\Sigma = (S_0, S, F)$ ist ein Tripel aus

- Menge von **Sorten** S,
- Menge von **Parametern** $S_0 \subseteq S$ und
- Menge von **Konstruktoren** F mit **Profilen** $c : a_1 \times \ldots \times a_n \to b$ mit $n \geq 0$, $a_1, \ldots, a_n \in S$, $b \in S \setminus S_0$.

Sortierte Terme

Kontext: $\Gamma = \{t_1 : a_1, \ldots, t_k : a_k\}$ mit $a_1, \ldots, a_k \in S$

$$\frac{}{\Gamma \vdash x : a}\ x : a \in \Gamma$$

$$\frac{\Gamma \vdash t_1 : a_1 \quad \ldots \quad \Gamma \vdash t_n : a_n}{\Gamma \vdash c(t_1, \ldots, t_n) : b}\ c : a_1 \times \ldots a_n \to b \in \Sigma$$

$$T_\Sigma(\Gamma)_a = \{t \mid \Gamma \vdash t : a\}$$

Mehrsortige Σ-Algebren:

- Mengen $\mathcal{M}[\![a]\!]$ für alle $a \in S$.
- Funktionen $\mathcal{M}[\![c]\!] : \mathcal{M}[\![a_1]\!] \times \ldots \times \mathcal{M}[\![a_n]\!] \to \mathcal{M}[\![b]\!]$ für alle $c : a_1 \times \ldots \times a_n \to b \in \Sigma$.

Mehrsortige Σ-Homomorphismen:

$g : \mathcal{M} \to \mathcal{N} = (g_a)_{a \in S}$ ist Familie von Abbildungen $g_a : \mathcal{M}[\![a]\!] \to \mathcal{N}[\![a]\!]$ mit $g_a = id$ für $a \in S_0$ und
$g_b(\mathcal{M}[\![c]\!](t_1, \ldots, t_n)) = \mathcal{N}[\![c]\!](g_{a_1}(t_1), \ldots, g_{a_n}(t_n))$ für jedes $c : a_1 \times \ldots \times a_n \to b \in \Sigma$.

Initiale mehrsortige Σ-Algebra

Gegeben: Menge V_a für jeden Parameter $a \in S_0$. Setze $\Gamma = \{x : a \mid a \in S_0, x \in V_a\}$. **Initiale mehrsortige Σ-Algebra** $\mathcal{M}$ ist definiert durch:

- $\mathcal{M}[\![a]\!] = T_\Sigma(\Gamma)_a$ für alle $a \in S$,
- $\mathcal{M}[\![c]\!](t_1, \ldots, t_n) = c(t_1, \ldots, t_n)$.

Initialität: Es existiert für alle mehrsortigen Σ-Algebren $\mathcal{N}$ mit $\mathcal{N}[\![a]\!] = V_a$ für alle $a \in S_0$ genau ein mehrsortiger Σ-Homomorphismus $g : \mathcal{M} \to \mathcal{N}$.

Initialität von $\mathcal{M}$ ermöglicht eindeutige Definition von primitiv-rekursiven Funktionen über mehrsortigen algebraischen Datentypen.

Übung 7 - Delay-Listen

data DelayList a, **Delay** a **where**
$Nil : () \rightarrow$ **DelayList** a
$Cons : a \rightarrow$ **Delay** $a \rightarrow$ **DelayList** a
$Now :$ **DelayList** $a \rightarrow$ **Delay** a
$Later : $ **Delay** $a \rightarrow$ **Delay** a

$countDelaysl\ Nil = 0$
$countDelaysl\ (Cons\ x\ d) = countDelaysd\ d$
$countDelaysd\ (Now\ dl) = 1 + (countDelaysl\ dl)$
$countDelaysd\ (Later\ d) = countDelaysd\ d$
$sumDelaysl\ Nil = 0$
$sumDelaysl\ (Cons\ x\ d) = sumDelaysd\ d$
$sumDelaysd\ (Now\ dl) = 1 + (sumDelaysl\ dl)$
$sumDelaysd\ (Later\ d) = 1 + (sumDelaysd\ d)$
$incDelaysl\ Nil = Nil$
$indDelaysl\ (Cons\ x\ d) = Cons\ x\ (incDelaysd\ d)$
$incDelaysd\ (Now\ dl) = Later\ (Now\ (incDelaysl\ dl))$
$incDelaysd\ (Later\ d) = Later\ (incDelaysd\ d)$

Übung 7 - Delay-Listen

Beweisen Sie mittels struktureller Induktion die folgenden Eigenschaften:

1. $\forall\, dl.\ \ countDelaysl\ \ (\ incDelaysl\ \ \ dl) = countDelaysl\ \ dl$
2. $\forall\, dl.\ \ sumDelaysl\ (\ incDelaysl\ \ \ dl) = (sumDelaysl\ \ dl) + (countDelaysl\ \ dl)$

Übung 8 - Ein koinduktives Animationsstudio

codata Animation where

$$
\begin{aligned}
\mathit{sprite} &: \textbf{Animation} \to \textbf{Sprite} \\
\mathit{advance} &: \textbf{Animation} \to \textbf{Animation} \\
\mathit{sprite}\ (\mathit{loop}\ (\mathit{Cons}\ s\ ss)) &= s \\
\mathit{advance}\ (\mathit{loop}\ (\mathit{Cons}\ s\ ss)) &= \mathit{loop}\ (\mathit{snoc}\ ss\ s) \\
\mathit{sprite}\ (\mathit{loop}\ \mathit{Nil}) &= \mathit{blankSprite} \\
\mathit{advance}\ (\mathit{loop}\ \mathit{Nil}) &= \mathit{loop}\ \mathit{Nil}
\end{aligned}
$$

1. Es sei $\mathit{walk\_right} = [s1, s2, s3, s4, s5, s6]$:

Was ist das Ergebnis der Auswertung des folgenden Terms?

$$\mathit{sprite}\ (\mathit{advance}\ (\mathit{advance}\ (\mathit{advance}\ (\mathit{loop}\ \mathit{walk\_right}\ ))))$$

Übung 8 - Ein koinduktives Animationsstudio

2. Definieren Sie eine korekursive Funktion

$$\mathit{delay} \ : \ \textbf{Animation} \to \textbf{Animation},$$

die die folgende Eigenschaft für alle $n \in \mathbb{N}$ erfüllt:

$$\mathit{sprite}\ (\mathit{advance}^n\ (\mathit{delay}\ a)) = \begin{cases} \mathit{sprite}\ a & \text{falls } n = 0 \\ \mathit{sprite}\ (\mathit{advance}^{n-1}\ a) & \text{falls } n > 0 \end{cases}$$

3. Definieren Sie eine korekursive Funktion
halfspeed : **Animation** $\to$ **Animation**, die jeden Frame verzögert.

4. Definieren Sie zwei Funktionen
doublespeed_e, *doublespeed_o* : **Animation** $\to$ **Animation** korekursiv, so dass *doublespeed_e* alle ungeraden und *doublespeed_o* alle geraden Frames überspringt.
Hinweis: Möglicherweise werden Sie eine Hilfsfunktion definieren müssen.

Übung 8 - Ein koinduktives Animationsstudio

5. Definieren Sie mittels Korekursion eine Funktion *prepend* : **List Sprite** $\rightarrow$ **Animation** $\rightarrow$ **Animation** welche eine einmalig abzuspielende Startsequenz vor einer Animation einfügt.
6. Wir nehmen an, dass eine Funktion *compatible* : **Sprite** $\rightarrow$ **Sprite** $\rightarrow$ **Bool** gegeben ist. Definieren Sie eine Funktion *transition* : **Animation** $\rightarrow$ **Animation** $\rightarrow$ **Animation**, so dass *transition a1 a2* eine Animation ist, die so lange *a1* abspielt, bis ein mit dem ersten Sprite von *a2* kompatibles Sprite erreicht wird, und sodann zu *a2* übergeht.