

9. Listen

9.1 Einführung Listentypen

Listen sind eine weitere zusammengesetzte Datenstruktur.

Alle Elemente einer Liste müssen denselben Typ haben. Die Werte eines Listentyps t list sind die endlichen Folgen von Werten des Typs t .

Induktive Definition von Werten des Typs t list:

1. nil ist ein Wert vom Typ t list, nämlich die leere Liste
2. Ist x ein Wert vom Typ t und ist xs eine Liste vom Typ t list, so ist auch $x::xs$ (gesprochen: x cons xs) eine Liste vom Typ t list.

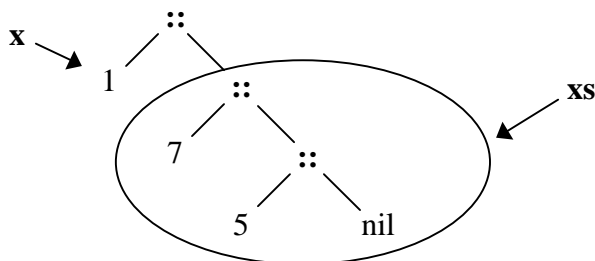


Abb. 9.1

3. Nichts sonst ist eine Liste vom Typ t list

list ist ein neuer **Typkonstruktor**
einstellig, **Postfix**-Notation

t list ist die Anwendung des Typkonstruktors list auf das Argument t .

nil und $::$ sind zwei neue **Wertkonstruktoren**.

- nil : 0-stellig, konstruiert die leere Liste
- $::$: 2-stellig mit
 - 1.Operand Komponententyp t
 - 2.Operand Listentyp t list

Listentyp t list ist **rekursiv**, da die Definition von t list wieder t list benutzt.

nil und $::$ sind **polymorph** im Komponententyp t des Listentyps t list.

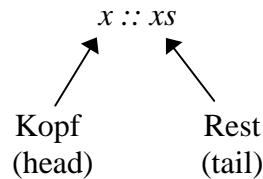
Jede Liste vom Typ t list hat die Form

$$(x_n :: \dots (x_2 :: (x_1 :: nil)))$$

$::$ klammert nach rechts, deswegen kann man die Klammern weglassen.

Abkürzende Schreibweise: $[x_n, \dots, x_1]$

9.2 Rechnen auf Listen



Rekursive Funktionen auf Listen nehmen ihre Listen auseinander in Kopf und Rest und sind rekursiv im Rest.

Konkatenation von zwei Listen: append
 Anzahl der Funktionsanwendungen für append (l,l')?
 length (l) + 1
 append ist in SML eingebaut als infix-operator @

Spiegelung einer Liste: nrev („naiv reverse“)
 Wieviele Funktionsanwendungen braucht nrev (l)?
 nrev [1,2,3]
 → nrev [2,3] @ [1]
 → (nrev [3] @ [2]) @ [1]
 → ((nrev [] @ [3]) @ [2]) @ [1]

Zahl der Funktionsanwendungen:

$$\begin{aligned}
 T(n) &= (n+1) + T(n-1) & n = \text{Länge der Listenargumente} \\
 &= (n+1) + n + T(n-2) \\
 &= \sum_{i=1}^{n+1} i = \frac{(n+2)(n+1)}{2}
 \end{aligned}$$

Größenordnung n^2

Effizientere Funktion zur Spiegelung von Listen: crev („clever reverse“)

Technik: akkumulierender Parameter

(l @ (x :: xs), nil)

(x :: xs, l^R)

Zahl der Funktionsanwendungen für crev (l)? length (l)

Korrektheit von crev

Beh: Für alle Werte l und a vom Typ t list gilt: crev (l,a) ↓ nrev(l) @ a

Folgerung: crev (l) = crev l (l,nil) ↓ nrev(l)

Beweis der Korrektheit von rekursiven Listenfunktionen: **Strukturelle Induktion**

Ind.Anf.: Zeige Behauptung für die leere Liste

Ind.Ann.: Behauptung gelte für Liste xs

Ind.Schritt: Zeige Behauptung für x :: xs

Beweis der Behauptung über `crev1` mit struktureller Induktion

$$\text{Ind.Anf.:} \quad \text{crev1}(\text{nil}, l) = l \stackrel{\substack{\text{nil neutrales} \\ \text{Element von @}}}{=} \text{nil} @ l \stackrel{\substack{\text{Def.von} \\ \text{nrev}}}{=} \text{nrev}(\text{nil}) @ l$$

$$\text{Ind.Ann.:} \quad \text{crev1}(xs, l) = \text{nrev}(xs) @ l$$

$$\begin{aligned} \text{Ind.Schritt:} \quad \text{crev1}(x :: xs, l) &= \text{crev1}(xs, x :: l) \stackrel{\text{Def.von @}}{=} \text{crev1}(xs, [x] @ l) \\ &= \text{nrev}(xs) @ ([x] @ l) \stackrel{\text{Ass.von @}}{=} (\text{nrev}(xs) @ [x]) @ l \\ &\stackrel{\text{Ind.Ann.}}{=} \text{nrev}(xs) @ ([x] @ l) \\ &\stackrel{\substack{\text{Def.von nrev} \\ \leftarrow}}{=} \text{nrev}(x :: xs) @ l \end{aligned}$$

q.e.d.