4. Übungsblatt zu Programmierung 1, WS 2012/13

Prof. Dr. Gert Smolka, Sigurd Schneider, B.Sc.
www.ps.uni-sb.de/courses/prog-ws12/

Lesen Sie im Buch: Kapitel 4

Aufgabe 3.19 Deklarieren Sie polymorphe Prozeduren wie folgt:

$$\begin{aligned} id: &\quad \forall\alpha.\ \alpha \to \alpha \\ com: &\quad \forall\alpha\ \beta\ \gamma.\ (\alpha \to \beta) \to (\beta \to \gamma) \to \alpha \to \gamma \\ cas: &\quad \forall\alpha\ \beta\ \gamma.\ (\alpha * \beta \to \gamma) \to \alpha \to \beta \to \gamma \\ car: &\quad \forall\alpha\ \beta\ \gamma.\ (\alpha \to \beta \to \gamma) \to \alpha * \beta \to \gamma \end{aligned}$$

Ihre Prozeduren sollen nicht rekursiv sein. Machen Sie sich klar, dass die angegebenen Typschemen das Verhalten der Prozeduren eindeutig festlegen.

Aufgabe 4.11 (Dezimaldarstellung) Die Dezimaldarstellung einer natürlichen Zahl ist die Liste ihrer Ziffern. Beispielsweise hat 7856 die Dezimaldarstellung $[7, 8, 5, 6]$.

a) Deklarieren Sie eine Prozedur $dec : int \to int\ list$, die die Dezimaldarstellung einer natürlichen Zahl liefert. Verwenden Sie *div* und *mod*.
b) Deklarieren Sie mithilfe von *foldl* eine Prozedur $num : int\ list \to int$, die zu einer Dezimaldarstellung die dargestellte Zahl liefert.

Aufgabe 4.12 Deklarieren Sie die Faltungsprozedur *foldr* mithilfe der Faltungsprozedur *foldl*. Verwenden Sie dabei keine weitere rekursive Hilfsprozedur.

Aufgabe 4.13 Deklarieren Sie die Faltungsprozedur *foldl* mithilfe der Faltungsprozedur *foldr*. Gehen Sie wie folgt vor:

a) Deklarieren Sie *append* mithilfe von *foldr*.
b) Deklarieren Sie *rev* mithilfe von *foldr* und *append*.
c) Deklarieren Sie *foldl* mithilfe von *foldr* und *rev*.
d) Deklarieren Sie *foldl* nur mithilfe von *foldr*.

Aufgabe 4.15 (Last) Deklarieren Sie eine Prozedur $last : \alpha\ list \to \alpha$, die das Element an der letzten Position einer Liste liefert. Wenn die Liste leer ist, soll die Ausnahme *Empty* geworfen werden.

Aufgabe 4.16 (Max) Schreiben Sie mit *foldl* eine Prozedur $max : int\ list \to int$, die das größte Element einer Liste liefert. Wenn die Liste leer ist, soll die Ausnahme *Empty* geworfen werden. Verwenden Sie die vordefinierte Prozedur $Int.max : int * int \to int$.

Aufgabe 4.17 (Take und Drop) Schreiben Sie zwei polymorphe Prozeduren *take* und *drop*, die gemäß $\alpha\ list * int \to \alpha\ list$ getypt sind und die folgende Spezifikation erfüllen:

a) $take(xs, n)$ liefert die ersten n Elemente der Liste xs. Falls $n < 0$ oder $|xs| < n$ gilt, soll die Ausnahme *Subscript* geworfen werden.

b) *drop*(xs, n) liefert die Liste, die man aus xs erhält, wenn man die ersten n Elemente weglässt. Falls $n < 0$ oder $|xs| < n$ gilt, soll die Ausnahme *Subscript* geworfen werden.

Hinweis: Verwenden Sie *orelse* und die Prozeduren *hd*, *tl* und *null*.

Aufgabe 4.19 Entscheiden Sie für jeden der folgenden Werte, ob er das Muster $(x,\ y :: \_ :: z,\ (u, 3))$ trifft. Geben Sie bei einem Treffer die Bindungen für die Variablen des Musters an.

a) $(7,\ [1],\ (3, 3))$
b) $([1, 2],\ [3, 4, 5],\ (11, 3))$

Aufgabe 4.21 Schreiben Sie eine Prozedur *size* : *string* → *int*, die die Länge eines Strings liefert (z.B. *size* "*hut*" = *3*).

Aufgabe 4.22 Schreiben Sie eine Prozedur *reverse* : *string* → *string*, die Strings reversiert (z.B. *reverse* "*hut*" = "*tuh*").

Aufgabe 4.23 Schreiben Sie eine Prozedur *isDigit* : *char* → *bool*, die mithilfe der Prozedur *ord* testet, ob ein Zeichen eine der Ziffern $0, \ldots, 9$ ist. Nützen Sie dabei aus, dass *ord* die Ziffern durch aufeinander folgende Zahlen darstellt.

Aufgabe 4.24 Schreiben Sie eine Prozedur *toInt* : *string* → *int*, die zu einem String, der nur aus Ziffern besteht, die durch ihn dargestellte Zahl liefert (z.B. *toInt* "*123*" = *123*). Falls der String Zeichen enthält, die keine Ziffern sind, soll die Ausnahme *Domain* geworfen werden.

Aufgabe 4.25 Schreiben Sie mithilfe der Prozeduren *explode* und *ord* eine Prozedur *less* : *string* → *string* → *bool*, die für zwei Strings s, s' dasselbe Ergebnis liefert wie $s < s'$.

Zusatzaufgabe Z4.1 Schreiben Sie eine Prozedur *enum*, sodass für $m \leq n$ gilt: *enum* $m\ n\ f = [f\,m, \ldots, f\,n]$. Für $m > n$ soll *enum* die leere Liste liefern.

Aufgabe 3.23 (Let und Abstraktionen) Dieter Schlau ist begeistert. Scheinbar kann man Let-Ausdrücke mit Val-Deklarationen auf Abstraktion und Applikation zurückführen:

$$\textit{let val } x = e \textit{ in } e' \textit{ end} \quad \rightsquigarrow \quad (\textit{fn } x\colon t \Rightarrow e')\,e$$

Auf der Heimfahrt von der Uni kommen ihm jedoch Zweifel an seiner Entdeckung, da ihm plötzlich einfällt, dass Argumentvariablen nicht polymorph getypt werden können. Können Sie ein Beispiel angeben, das zeigt, dass Dieters Zweifel berechtigt sind? Hilfe: Geben Sie einen semantisch zulässigen Let-Ausdruck an, dessen Übersetzung gemäß des obigen Schemas scheitert, da Sie keinen passenden Typ t für die Argumentvariable x finden können.

Aufgabe 4.2 Betrachten Sie den Ausdruck 1::2::*nil*@3::4::*nil*.

a) Geben Sie die Baumdarstellung des Ausdrucks an.
b) Geben Sie die Baumdarstellung der beschriebenen Liste an.
c) Geben Sie die beschriebene Liste mit „[...]“ an.