Technische Universität München
Fakultät für Informatik
R. Kuntschke, I. Nagy,
Dr. M. Schneider

SS 2006
Lösungsvorschläge zu Blatt 4
2. Juni 2006

Übungen zu Einführung in die Informatik I

Aufgabe 13 Listen in OCaml (Lösungsvorschlag)

a)
```
# type 'a myList = Empty | Elem of 'a * 'a myList;;
type 'a myList = Empty | Elem of 'a * 'a myList
```

b)
```
# let prepend a l = Elem(a,l);;
val prepend : 'a -> 'a myList -> 'a myList = <fun>
# let test = prepend 'a' (prepend 'h' (prepend 'a'  Empty));;
- : char myList =
Elem ('a', Elem ('h', Elem ('a', Empty)))
```

Alternativ:

```
# let test = prepend 'a' Empty;;
val test : char myList = Elem ('a', Empty)
# let test = prepend 'h' test;;
val test : char myList = Elem ('h', Elem ('a', Empty))
# let test = prepend 'a' test;;
val test : char myList = Elem ('a', Elem ('h', Elem ('a', Empty)))
```

c)
```
# let rec append a l =
  match l with
    Empty        -> Elem(a,Empty)
  | Elem(b,tail) -> Elem(b, (append a tail));;
val append : 'a -> 'a myList -> 'a myList = <fun>
# append '!' test;;
- : char myList = Elem ('a', Elem ('h', Elem ('a', Elem ('!', Empty))))
```

d)
```
# let rec iselem a l =
  match l with
  Empty                  -> false
  | Elem(b,tail) when a=b -> true
  | Elem(b,tail)          -> iselem a tail;;
val iselem : 'a -> 'a myList -> bool = <fun>
# iselem 'x' test;;
- : bool = false
# iselem 'h' test;;
- : bool = true
# iselem 'a' test;;
- : bool = true
```

e)
```
# let rec insert a l =
  match l with
    Empty                 -> Elem(a,Empty)
  | Elem(b,tail) when a<b -> Elem(b, (insert a tail))
  | Elem(b,tail)          -> Elem(a, (prepend b tail));;
val insert : 'a -> 'a myList -> 'a myList = <fun>
# let test = insert 23 Empty;;
val test : int myList = Elem (23, Empty)
# let test = insert 12 test;;
val test : int myList = Elem (23, Elem (12, Empty))
# let test = insert 42 test;;
val test : int myList = Elem (42, Elem (23, Elem (12, Empty)))
# let test = insert 24 test;;
val test : int myList = Elem (42, Elem (24, Elem (23, Elem (12, Empty))))
# let test = insert 1 test;;
val test : int myList =
  Elem (42, Elem (24, Elem (23, Elem (12, Elem (1, Empty)))))
# let test = insert 23 test;;
val test : int myList =
  Elem (42, Elem (24, Elem (23, Elem (23, Elem (12, Elem (1, Empty))))))
```

Aufgabe 14 Das Prinzip der Induktion (Lösungsvorschlag)

Zur vollständigen Induktion gibt es mehrere Varianten. In dieser Aufgabe werden Sie eine erste Form der Induktion kennenlernen:
Das Prinzip: Es sei A(n) eine Aussage über einer natürlichen Zahl n, dann bedeutet das Prinzip der vollständigen Induktion folgendes:

- Gilt für ein $n_0 \in \mathbb{N}$ die Aussage A(n) (**Induktionsanfang**)
- und folgt aus der Gültigkeit der Aussage A(n) die Gültigkeit von A(n+1) (**Induktionsschritt**)
- dann gilt A(n) für alle $n \in \mathbb{N}$ mit $n \geq n_0$.

Wir beweisen nun die Aussage $A(x) = (\texttt{gerade}\ (x) \stackrel{?}{=} \texttt{gerade}'\ (x))$

(i) **Induktionsanfang:** Es sei $x_0 = 0$. Dann folgt:

$$\begin{aligned} \texttt{gerade}\ (x_0) &= (\mathrm{mod}(0,2) \stackrel{?}{=} 0) \\ &= (0 \stackrel{?}{=} 0) \\ &= \mathit{True}. \end{aligned}$$

Andererseits ist $\texttt{gerade}'\ (x_0) = \mathit{True}$ gemäß Defintion und somit gilt $A(x_0)$.

(ii) **Induktionsschritt:** Nun ist zu zeigen, dass aus der Gültigkeit von $A(x)$, die Gültigkeit von $A(x+1)$ folgt. Gelte nun also $A(x)$ für ein $x \geq x_0$ (**Induktionsannahme**); dann ist zu zeigen, dass daraus $(\texttt{gerade}\ (x+1) \stackrel{?}{=} \texttt{gerade}'\ (x+1))$ folgt.

Wir betrachten `gerade'` $(x+1)$:

$$\begin{array}{rcll}
\texttt{gerade}'\,(x+1) & = & \neg(\texttt{gerade}'\,(x)) & \text{Definition von } \texttt{gerade' (x)} \\
& = & \neg(\texttt{gerade}\,(x)) & \text{Induktionsannahme} \\
& = & \neg(\mathrm{mod}(x,2) \stackrel{?}{=} 0) & \text{Definition von } \texttt{gerade(x)} \\
& = & (\mathrm{mod}(x,2) \stackrel{?}{=} 1) & \text{Rest bei Division durch 2 ist entweder 0 oder 1} \\
& = & (\mathrm{mod}(x+1,2) \stackrel{?}{=} 0) & \\
& = & \texttt{gerade}\,(x+1) &
\end{array}$$

Somit folgt aus $A(x)$ die Gültigkeit von $A(x+1)$ und die Äquivalenz von `gerade` (x) und `gerade'` (x) ist gezeigt.

Aufgabe 15 Induktion über natürlichen Zahlen (Lösungsvorschlag) (10 Punkte)

a) Bei dieser Induktion sind k Induktionsanfänge durchzuführen; der Induktionsschritt erfolgt demzufolge mit $n+k$.

Induktionsanfang: Bei den Induktionsanfänfangen betrachten wir den Fall $n = 0$ und $0 < n < k$:

- $n = 0$; hierbei gilt: $istTeiler(k,0) = (\exists m \in \mathbb{N}_0 : m \cdot k = 0)$; mit $m = 0$ folgt $istTeiler(k,0) = true$. Andererseits ist $istTeiler_{Imp}(k,0) = true$ per Definition und die Gleichheit ist für $n = 0$ gezeigt.
- $0 < n < k$. In diesem Fall gibt es kein m, so dass das Prädikat $\exists m \in \mathbb{N}_0 : m \cdot k = n$ erfüllt ist, somit gilt $istTeiler(k,n) = false$, wie auch $istTeiler_{Imp}(n,k) = false$ ist und die Gleichheit der beiden Funktionen gilt auch hier.

Induktionsschritt: Induktionshypothese ist, dass die Gleichheit der beiden Funktionen für alle $n' < n+k$ gezeigt ist. Wir betrachten

$$\begin{array}{rcl}
istTeiler_{Imp}(k,n+k) & = & istTeiler_{Imp}(k,n) \\
& = & istTeiler(k,n) \\
& = & (\exists m \in \mathbb{N}_0 : m \cdot k = n) \\
& = & (\exists m \in \mathbb{N}_0 : m \cdot k + k = n + k) \\
& = & (\exists m \in \mathbb{N}_0 : (m+1) \cdot k = n + k) \\
& = & (\exists m' \in \mathbb{N}_0 : m' \cdot k = n + k) \\
& = & istTeiler(k,n+k)
\end{array}$$

Somit ist die Gleichheit der beiden Funktionen gezeigt.

b) Eine rekursive Funktion, die die Anzahl der Aufrufe der Funktion `istTeiler` zählt, lautet:

```
let rec countCalls (n, k) = match (n, k) with
                  | (n, k) when (n = 0) || (n > 0 && n < k) -> 1
                  | (n, k) -> 1 + countCalls (k, n-k) ;;
```

Aufgabe 16 Induktion: `take` und `drop` (Lösungsvorschlag)

a) Eine mögliche Lösung könnte folgendermaßen aussehen:

```
let list = [1; 2; 3; 4; 5; 6; 7; 8; 9; 0];;

let take(m, ls) =
  let rec take_em(count, list, taken) =
    if count = 0 then taken
    else
      match list with
      | [] -> taken
      | head::tail -> take_em(count - 1, tail, taken@[head]) in
  take_em(m, ls, []);;

take(3, list);;

let drop(n, ls) =
  let rec drop_em(count, list) =
    if count = 0 then list
    else
      match list with
      | [] -> []
      | head::tail -> drop_em(count - 1, tail) in
  drop_em(n, ls);;

drop(4, list);;
```

b) Wir beweisen das Prädikat $\forall\, \mathtt{n} \in \mathbb{N}_0,\ \mathtt{xs} \in [a] : \mathtt{concat}(\mathtt{take\ n\ xs},\ \mathtt{drop\ n\ xs}) = \mathtt{xs}$ mittels vollständiger Induktion über n.

Induktionsanfang: $n = 0$

$$\begin{array}{lcll} \mathtt{concat}(\mathtt{take\ n\ xs},\ \mathtt{drop\ n\ xs}) & = & \mathtt{concat}(\mathtt{take\ 0\ xs},\ \mathtt{drop\ 0\ xs}) & \\ & = & \mathtt{concat}([],\ \mathtt{xs}) & \text{Def. } \mathtt{take}, \mathtt{drop} \\ & = & \mathtt{xs} & \text{Def. } \mathtt{concat} \end{array}$$

Induktionsschritt: $n = m + 1$

$$\begin{array}{ll} \forall\, \mathtt{zs} \in [a] : \mathtt{concat}(\mathtt{take\ m\ zs},\ \mathtt{drop\ m\ zs}) = \mathtt{zs} & \text{Induktionshypothese} \\ \mathtt{concat}(\mathtt{take\ n\ xs},\ \mathtt{drop\ n\ xs}) = \mathtt{concat}(\mathtt{take\ m} + 1\ \mathtt{xs},\ \mathtt{drop\ m} + 1\ \mathtt{xs}) & \text{Induktionsannahme} \end{array}$$

- `xs = []`

$$\begin{array}{lcl} \mathtt{concat}(\mathtt{take\ m} + 1\ \mathtt{xs},\ \mathtt{drop\ m} + 1\ \mathtt{xs}) & = & \mathtt{concat}(\mathtt{take\ m} + 1\ [],\ \mathtt{drop\ m} + 1\ []) \\ & = & \mathtt{concat}([], []) \\ & = & [] \\ & = & \mathtt{xs} \end{array}$$

- `xs = y::ys`

$$\begin{array}{lcl} \mathtt{concat}(\mathtt{take\ m} + 1\ \mathtt{xs},\ \mathtt{drop\ m} + 1\ \mathtt{xs}) & = & \mathtt{concat}(\mathtt{take\ m} + 1\ \mathtt{y} :: \mathtt{ys},\ \mathtt{drop\ m} + 1\ \mathtt{y} :: \mathtt{ys}) \\ & = & \mathtt{concat}(\mathtt{y} :: \mathtt{take\ m\ ys},\ \mathtt{drop\ m\ ys}) \\ & = & \mathtt{y} :: \mathtt{concat}(\mathtt{take\ m\ ys},\ \mathtt{drop\ m\ ys}) \\ & = & \mathtt{y} :: \mathtt{ys} \\ & = & \mathtt{xs} \end{array}$$