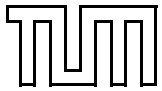
TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK



WS 11/12

Einführung in die Informatik II **Übungsblatt 5**

Univ.-Prof. Dr. Andrey Rybalchenko, M.Sc. Ruslán Ledesma Garza 29.11.2011

Dieses Blatt behandelt Kapitel 3.4, 3.6, 3.13, 4.1 und 4.2 aus dem Buch zur Vorlesung. Lesen Sie diese Kapitel!

Aufgabe 3.24

Deklarieren Sie eine Prozedur `first : int → (int → bool) → int`, die zu x und p die kleinste Zahl $y \geq x$ mit `p y = true` liefert. Verzichten Sie dabei vollständig auf Typangaben.

Lösungsvorschlag 3.24:

```
fun first s p = if p s then s else first (s+1) p
```

Aufgabe 3.25

Geben Sie die Typschemen an, mit denen die Bezeichner `p` und `q` des folgenden Programms typisiert werden.

```
fun p f (x,y) = f x y
fun q f g x = g (f x)
```

Lösungsvorschlag 3.25:

p : $\forall\alpha\beta\gamma.(\alpha \to \beta \to \gamma) \to (\alpha * \beta) \to \gamma$

q : $\forall\alpha\beta\gamma.(\alpha \to \beta) \to (\beta \to \gamma) \to \alpha \to \gamma$

Aufgabe 3.26

Geben Sie Deklarationen an, die monomorph getypte Bezeichner wie folgt deklarieren:

a) `int * unit * bool`

b) `unit * (int * unit) * (real * unit)`

c) `int → int`

d) `int * bool → int`

e) `int → real`

f) `int → real → real`

g) `(int→int)→bool`

Verzichten Sie dabei auf explizite Typangaben und verwenden Sie keine Operator- und Prozeduranwendungen. Hinweis: Für einige der Deklarationen ist die Verwendung eines Konditionals essenti- ell. Die Typregel für Konditionale verlangt, dass die Konsequenz und die Alternative den gleichen Typ haben (siehe § 2.6). Außerdem ist für einige der Deklarationen die Verwen- dung von Tupeln und Projektionen erforderlich, um Werte vergessen zu können, die nur zur Steuerung der Typinferenz konstruiert wurden.

Lösungsvorschlag 3.26:

a) **val** `v = (10,(),true)`

b) **val** `v = ((),(10,()),(1.0,()))`

c) **fun** `f i =` **if** `true` **then** `i` **else** `10`

d) **fun** `f (i,b) =` **if** `b` **then** `i` **else** `1`

e) **fun** `f i =` **let val** `v=` **if** `true` **then** `i` **else** `10` **in if** `false` **then** `1.0` **else** `0.0` **end**

f) **fun** `f i r =` **let val** `v =` **if** `true` **then** `i` **else** `10` **val** `v2=` **if** `false` **then** `1.0` **else** `r` **in** `v2` **end**

g) **fun** `f g =` **let val** `v =` **if** `true` **then** `g` **else** `(`**fn** `x` ⇒ **if** `true` **then** `x` **else** `1)` **in** `true` **end**

Aufgabe 3.27 **

Im Zusammenhang mit fehlenden Typangaben kann die Verwendung von Projektionen problematisch sein. Beispielsweise kann Typinferenz das Programm **fun** `f x = #1x` nicht typisieren. Können Sie erklären, warum das so ist? **Hinweis:** Lesen Sie das Kapitel 2.8.

Lösungsvorschlag 3.27 **:

Die typisierung des programs ist statisch (Kap 2.8 unten bei statischer semantik), statisch lässt sich aber nicht feststellen ob der ausdruck x ein tupel mit mindestens 1 komponente ist. Dies würde nur funktionieren wenn der typ von x zur Laufzeit überprüft würde.

Aufgabe 3.36 Bitte lessen Sie §3.13. Deklarieren Sie mithilfe der Prozedur `iterup` eine Prozedur

a) `power`, die zu x und n die Potenz x^n liefert.

b) `fac`, die zu $n \geq 0$ die n-te Fakultät $n!$ liefert.

c) `sum`, die zu f und n die Summe $0 + f\,1 + \ldots + f\,n$ liefert.

d) `iter'`, die zu n, s und f dasselbe Ergegnis liefert wie `iter n s f`.

Lösungsvorschlag 3.36:

```
fun power x n = iterup 1 n 1 (fn (m, s) ⇒ s * x);
fun fac n = iterup 1 n 1 (fn (m, s) ⇒ s * m);
fun sum f n = iterup 1 n 0 (fn (m,s) ⇒ s + f m);
fun iter' n s f = iterup 1 n s (fn (m,s) ⇒ f s);
```

Aufgabe 3.37 Bitte lessen Sie §3.13. Deklarieren Sie mithilfe der Prozedur `iter` eine Prozedur

a) `iterup'`, die zu `m`, `n`, `s` und `f` dasselbe Ergebnis wie `iterup m n s f` liefert.

b) `iterdn'`, die zu `n`, `m`, `s` und `f` dasselbe Ergebnis wie `iterdn n m s f` liefert.

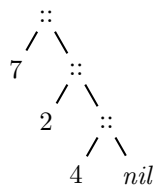
Lösungsvorschlag 3.37:

```
fun iterup' m n s f = #2 (iter (n-m+1) (m,s) (fn (x,s) ⇒ (x+1,f (x,s))))
fun iterdn' n m s f = #2 (iter (n-m+1) (n,s) (fn (x,s) ⇒ (x-1,f (x,s))))
```

Aufgabe 4.1 Geben Sie einen Ausdruck an, der die Liste `[7, 2, 4]` klammerfrei mit Cons und *nil* beschreibt. Geben Sie die Baumdarstellung ihres Ausdrucks an. Unterscheidet sich die Baumdarstellung des Ausdrucks von der Baumdarstellung der Liste?

Lösungsvorschlag 4.1:

```
7 :: 2 :: 4 :: nil
```



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

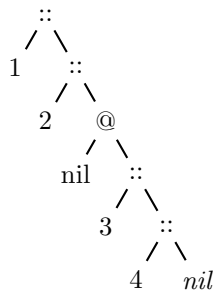
a) Geben Sie die Baumdarstellung des Ausdrucks an.

b) Geben Sie die Baumdarstellung der beschriebenen Liste an.

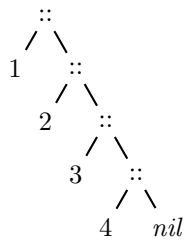
c) Geben Sie die beschriebene Liste mit "[...]" an.

Lösungsvorschlag 4.2:

a)



b)



c) `[1, 2, 3, 4]`

Aufgabe 4.3 Macht es für die dargestellten Listen einen Unterschied, wie die folgenden Ausdrücke geklammert sind?

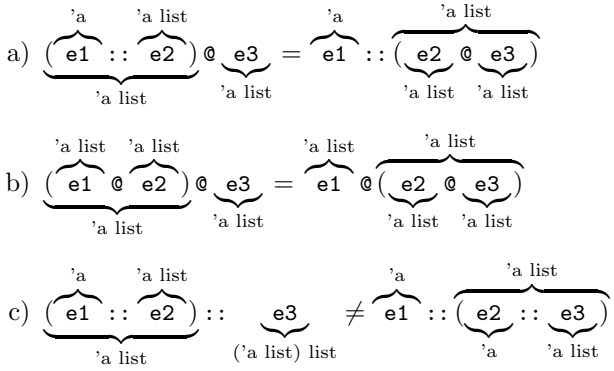
a) `(e1::e2)@e3` oder `e1::(e2@e3)`.

b) `(e1@e2)@e3` oder `e1@(e2@e3)` .

c) `(e1::e2)::e3` oder `e1::(e2::e3)`.

Lösungsvorschlag 4.3:

```
op :: : 'a      * 'a list → 'a list
op @  : 'a list * 'a list → 'a list
```



Aufgabe 4.4 (Enum) Schreiben Sie mithilfe der Prozedur `iterdn` (§3.13) eine Prozedur `enum:int→int→ int list`, die zu zwei Zahlen $m \leq n$ die Liste $[m, \ldots, n]$ liefert. Beispielsweise soll `enum 3 6 = [3, 4, 5, 6]` gelten. Für $m > n$ soll `enum` die leere Liste liefern.

Lösungsvorschlag 4.4:

```
fun enum n m = iterdn m n nil (fn (n,x) ⇒ n::x)
```