Ludwig-Maximilians-Universität München WS 2005/2006
Institut für Informatik Blatt 7
Prof. Dr. F. Kröger, Dr. M. Hölzl, F. Hacklinger

Übungen zu Informatik I (Lösungsvorschlag)

Aufgabe 7-1 Multiplikation in curried Form (keine Abgabe)

a) Eine mögliche Herleitung ist die folgende:

(1)	$\{x : \mathbf{int}\} \rhd x : \mathbf{int}$	Typaxiom
(2)	$\{y : \mathbf{int}\} \rhd y : \mathbf{int}$	Typaxiom
(3)	$\emptyset \rhd * : \mathbf{int} * \mathbf{int} \to \mathbf{int}$	Typaxiom
(4)	$\{x : \mathbf{int}, y : \mathbf{int}\} \rhd x * y : \mathbf{int}$	(R2) mit Prämissen (1), (2) und (3)
(5)	$\{x : \mathbf{int}\} \rhd \mathbf{fn}\ y \Rightarrow x * y : \mathbf{int} \to \mathbf{int}$	(R4) mit Prämisse (4)
(6)	$\emptyset \rhd \mathbf{fn}\ x \Rightarrow \mathbf{fn}\ y \Rightarrow x * y : \mathbf{int} \to \mathbf{int} \to \mathbf{int}$	(R4) mit Prämisse (5)

b) Um die gewünschte Herleitung zu erhalten, verlängern wir die Herleitung aus Teilaufgabe (a):

(7)	$\emptyset \rhd 5 : \mathbf{int}$	Typaxiom
(8)	$\emptyset \rhd (\mathbf{fn}\ x \Rightarrow \mathbf{fn}\ y \Rightarrow x * y)\ 5 : \mathbf{int} \to \mathbf{int}$	(R5) mit Prämissen (6) und (7)

Aufgabe 7-2 Ausnahmebehandlung (keine Abgabe)

a)
```
(* Nochmals die Fakultaetsfunktion aus der Vorlesung *)
fun fak n = if n=0 then 1 else n*fak(n-1)

(* Die Funktion safe_fak *)
fun safe_fak n = fak n handle overflow => ~1;
```

b)
```
fun fast_exp (n, m) =
    if m = 0
    then 1
    else if m mod 2 = 0
    then let val k = fast_exp(n, m div 2)
         in k * k
         end
    else n * fast_exp(n, m-1);;

fun safe_exp(n, m) =
    fast_exp(n, m)
    handle overflow => ~1;;
```

Die im Folgenden definierte Funktion *bad_exp* führt nur dann eine Fehlerbehandlung durch, wenn der Überlauf bei der Berechnung von $n * \mathit{bad\_exp}(n, m-1)$ auftritt. Außerdem rechnet die Funktion nach dem Auftreten einer Ausnahme mit dem Wert -1 weiter: z.B. $\mathit{bad\_exp}(7, 11) = -1$, $\mathit{bad\_exp}(7, 22) = 1$, $\mathit{bad\_exp}(7, 23) = 7$.

```
fun bad_exp(n, m) = if m = 0
                    then 1
                    else if m mod 2 = 0
                         then let val k = bad_exp(n, m div 2)
                              in k * k
                              end
                         else n * bad_exp(n, m-1)
                    handle overflow => ~1;;
```

Aufgabe 7-3 **Komplexe Zahlen** (keine Abgabe)

```
type complex = {realteil:real, imgteil:real};

fun addcomplex(a:complex, b:complex):complex =
   {realteil = #realteil(a) + #realteil(b), imgteil = #imgteil(a) + #imgteil(b)};

fun multcomplex(a:complex, b:complex):complex =
    { realteil = #realteil(a) * #realteil(b)- #imgteil(a) * #imgteil(b),
      imgteil = #imgteil(a) * #realteil(b) + #realteil(a) * #imgteil(b)};

(*
val a = {realteil=1.0, imgteil=2.3};
val b = {realteil=4.1, imgteil=2.1};
addcomplex(a,a);
addcomplex(a,b);
multcomplex(a,b);
multcomplex(b,b);
*)
```

Aufgabe 7-4 **Typisierung mit lokalen Konstanten** (4 Punkte)

Wir verwenden neben der Typisierungsregel (R6) auch die Regel (R5) und erhalten:

(1)	$\{x : \mathbf{int}\} \rhd x : \mathbf{int}$	Typaxiom
(2)	$\emptyset \rhd 1 : \mathbf{int}$	Typaxiom
(3)	$\emptyset \rhd + : \mathbf{int} * \mathbf{int} \to \mathbf{int}$	Typaxiom
(4)	$\{x : \mathbf{int}\} \rhd x + 1 : \mathbf{int}$	(R2) mit Prämissen (1), (2) und (3)
(5)	$\{x2 : \mathbf{int}\} \rhd x2 : \mathbf{int}$	Typaxiom
(6)	$\{x : \mathbf{int}, x2 : \mathbf{int}\} \rhd x2 + x + 1 : \mathbf{int}$	(R2) mit Prämissen (4), (5) und (2)
(7)	$\{x : \mathbf{int}\} \rhd \mathbf{fn}\ x2 \Rightarrow x2 + x + 1 : \mathbf{int} \to \mathbf{int}$	(R4) mit Prämisse (6)
(8)	$\emptyset \rhd 2 : \mathbf{int}$	Typaxiom
(9)	$\emptyset \rhd * : \mathbf{int} * \mathbf{int} \to \mathbf{int}$	Typaxiom
(10)	$\{x : \mathbf{int}\} \rhd x * 2 : \mathbf{int}$	(R2) mit Prämissen (1), (8) und (9)
(11)	$\{x : \mathbf{int}\} \rhd (\mathbf{fn}\ x2 \Rightarrow x2 + x + 1)(x * 2) : \mathbf{int}$	(R5) mit Prämissen (7) und (10)
(12)	$\{x : \mathbf{int}\} \rhd \mathbf{let\ val}\ x2 = x * 2\ \mathbf{in}\ x2 + x + 1\ \mathbf{end} : \mathbf{int}$	(R6) mit Prämisse (11)

Hinweis: Die in (R6) angegebene Beziehung zwischen Funktionsabstraktion und lokaler Bindung gilt nicht nur für die Typisierung, es ist auch $W((\mathbf{fn}\ x \Rightarrow s)(t)) = W(\mathbf{let\ val}\ x = t\ \mathbf{in}\ s\ \mathbf{end})$.

Aufgabe 7-5 **Mustervergleich** (4 Punkte)

a) Die gesuchte Funktion *qzt_pm* ist wie folgt mit Mustervergleich definierbar:

```
fun qzt_pm (n, 0) = n = 0
  | qzt_pm (n, k) = n = k*k orelse qzt_pm (n, k-1)
```

b) Die Funktion *teilersumme_pm* ist wie folgt definiert:

```
fun teilersumme_pm(n, 0) = 0
  | teilersumme_pm(n, k) = (if n mod k = 0 then k else 0) + teilersumme_pm(n, k-1);
```

c) Die gewünschte Funktion *hatteiler_pm* kann man wie folgt mit Mustervergleich definieren:

```
fun hatteiler_pm (n, 1) = false
  | hatteiler_pm (n, k) = n mod k = 0 orelse hatteiler_pm (n, k-1)
```

Aufgabe 7-6 **Brüche** (4 Punkte)

```
(*a*)
type bruch = {zaehler:int, nenner:int};

(*b*)
fun multfrac(a:bruch, b:bruch) =
    {zaehler = #zaehler(a)* #zaehler(b), nenner = #nenner(a) * #nenner(b)};

(*c*)
fun addfrac(a:bruch, b:bruch) =
    {zaehler = #zaehler(a) * #nenner(b) + #zaehler(b) * #nenner(a),
                            nenner = #nenner(a) * #nenner(b)};

(*d*)
fun divfrac(a:bruch, b:bruch) = multfrac(a, invert(b))
and invert(c:bruch) = {zaehler = #nenner(c), nenner = #zaehler(c)};

(*Aufrufe*)
val  a = {zaehler=1, nenner=2};
multfrac(a,a);
addfrac(a,multfrac(a,a));
divfrac(a,a);
```