



**Programmierung:
Musterlösung zum 11. Übungsblatt**

Prof. Gert Smolka und Thorsten Brunklaus

Aufgabe 11.1: F/FR Ausdrücke (1+1+1+1)

(a) 1 1

(b) (i)

$\text{rec } \bar{f}_1 (\bar{n}_1) : \text{int} \Rightarrow \text{if } n_1 \leq 1 \text{ then } 1 \text{ else } n_1 * \bar{f}_1 (n_1 - 1)$

(ii)

$\text{rec } \bar{x}_1 (\bar{x}_2) : \text{int} \rightarrow \text{int} \Rightarrow \text{fn } \bar{y}_1 \Rightarrow z x_2 y_1$

(c) (i)

```
val alt = Op(Id "n", Mul, App(Id "f", Op(Id "n", Sub, Con (IC 1))))
val body = If(Op(Id "n", Leq, Con (IC 1)), Con (IC 1), alt)
val r1 = Rec("f", "n", Int, Int, body)
```

(ii)

```
val body = Abs("y", Int, App(App(Id "z", Id "x"), Id "y"))
val r2 = Rec("x", "x", Int, Arrow (Int, Int), body)
```

(d)

$((\text{fn } x : \text{int} \Rightarrow x) 5) + ((\text{fn } x : \text{int} \Rightarrow x) 3)$

Aufgabe 11.2: Erweiterung von F um Andalso (4+4+4)

$$\frac{T \vdash e_1 \Rightarrow \text{bool} \quad T \vdash e_2 \Rightarrow \text{bool}}{T \vdash e_1 \text{ andalso } e_2 \Rightarrow \text{bool}}$$

$$\frac{V \vdash e_1 \Rightarrow 1 \quad V \vdash e_2 \Rightarrow v}{V \vdash e_1 \text{ andalso } e_2 \Rightarrow v}$$

$$\frac{V \vdash e_1 \Rightarrow 0}{V \vdash e_1 \text{ andalso } e_2 \Rightarrow 0}$$

$$\frac{}{\text{false andalso } e_2 \rightarrow \text{false}}$$

$$\frac{}{\text{true andalso } e_2 \rightarrow e_2}$$

$$\frac{e_1 \rightarrow e'_1}{e_1 \text{ andalso } e_2 \rightarrow e'_1 \text{ andalso } e_2}$$

Aufgabe 11.3: Freie Bezeichner (6)

```
fun freeIds' xs (Con c)          = []
| freeIds' xs (Id s)            = if List.exists (fn x => x=s) xs
                                then [] else [s]
| freeIds' xs (Op(e1,ops,e2))   = freeIds' xs e1 @ freeIds' xs e2
| freeIds' xs (If(e1,e2,e3))    = freeIds' xs e1 @ freeIds' xs e2
                                @ freeIds' xs e3
| freeIds' xs (Abs(s,t,e))      = freeIds' (s::xs) e
| freeIds' xs (App(e1,e2))      = freeIds' xs e1 @ freeIds' xs e2
| freeIds' xs (Rec(s,s',t',t,e)) = freeIds' (s::s'::xs) e

fun freeIds e = freeIds' [] e
```

Aufgabe 11.4: F mit Rekursion (9)

```
| elab T (Rec(s,s',t',t,e)) =
  let
    val T' = insert(s',t',insert(s,Arrow(t',t),T))
  in
    if elab T' e = t then Arrow(t',t)
    else raise Error "ill-typed recursive abstraction"
  end

| eval V (Rec(s,s',t',t,e)) = RProc(s,s',e,V)
| eval V (App(e1,e2)) =
  (case (eval V e1, eval V e2) of
    (Proc(s,e,V'), v)      =>
      eval (insert(s, v, 'V')) e
  | (p as RProc(s,s',e,V'), v') =>
      eval (insert(s', v', insert(s, p, V')) e
  | _                      => raise Error "type error")
```

Aufgabe 11.5: Einscrist-Semantik (3+4+4+4+4)

```

fun canonical (Con _) = true
  | canonical (Abs _) = true
  | canonical _ = false

fun subst (Con c)          e' s' = Con c
  | subst (Id s)           e' s' = if s=s' then e' else Id s
  | subst (Op(e1,ops,e2))  e' s' = Op(subst e1 e' s', ops, subst e2 e' s')
  | subst (If(e1,e2,e3))   e' s' = If(subst e1 e' s', subst e2 e' s', subst e3 e' s')
  | subst (Abs(s,t,e))      e' s' = if s=s' then Abs(s,t,e)
                                   else Abs(s,t, subst e e' s')
  | subst (App(e1,e2))      e' s' = App(subst e1 e' s', subst e2 e' s')
  | subst (Rec(s,s',t',t,e)) e' s'' = if s=s'' orelse s'=s'' then Rec(s,s',t',t,e)
                                   else Rec(s,s',t',t, subst e e' s'')

fun reduceOp (IC x) Add (IC x') = Con(IC(x+x'))
  | reduceOp (IC x) Sub (IC x') = Con(IC(x-x'))
  | reduceOp (IC x) Mul (IC x') = Con(IC(x*x'))
  | reduceOp (IC x) Leq (IC x') = Con(if x<=x' then True else False)
  | reduceOp _ _ _ = raise Error "Impossible"

fun reduce (Op(Con x1, ops, Con x2)) = reduceOp x1 ops x2
  | reduce (Op(e1, ops, e2)) = if canonical e1 then Op(e1, ops, reduce e2)
                                else Op(reduce e1, ops, e2)
  | reduce (If(Con True, e2, e3)) = e2
  | reduce (If(Con False, e2, e3)) = e3
  | reduce (If(e1, e2, e3)) = If(reduce e1, e2, e3)
  | reduce (App(Abs(s,t,e), e')) = if canonical e' then subst e e' s
                                   else App(Abs(s,t,e), reduce e')
  | reduce (App(e1,e2)) = App(reduce e1, e2)
  | reduce (e as Rec(s,s',t',t,e')) = Abs(s',t', subst e' e s)
  | reduce _ = raise Error "Impossible"

fun normalize e = if canonical e then e else normalize(reduce e)

```

Aufgabe 11.6: Challenge: Fakultät ohne Rekursion

```

(fn x => x x)
(fn f => fn n =>
  if n<=1 then 1 else n * f f (n-1))

```