TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

WS 11/12

Einführung in die Informatik 2 **Klausur**
Prof. Dr. Andrey Rybalchenko, M.Sc. Ruslán Ledesma Garza 11.02.2011

Name	Vorname	Studiengang	Matrikelnummer
Hörsaal	Reihe	Sitzplatz	Unterschrift

Allgemeine Hinweise:

- **Bitte füllen Sie die oben angegebenen Felder vollständig aus und unterschreiben Sie!**
- **Schreiben Sie nicht mit Bleistift oder in roter/grüner Farbe!**
- **Die Arbeitszeit beträgt 105 Minuten.**
- **Prüfen Sie, ob Sie alle 13 Seiten erhalten haben.**
- **In dieser Klausur können Sie insgesamt 105 Punkte erreichen. Zum Bestehen werden 42 Punkte benötigt.**
- **Als Hilfsmittel ist nur ein beidseitig handbeschriebenes DinA4-Blatt zugelassen. Das Merkblatt mit nützlichen Prozeduren ist als letztes Blatt an die Klausur angeheftet.**

Aufgabe 1 [20 Punkte] **Programauswertung**

1.1 Werte

Geben Sie den Wert der folgenden Ausdrücke an.

```
(* a *)
let fun p x = x mod 2 <> 0 in
  p 9 andalso p 3
end                                        Wert = true

(* b *)
map
  (fn x => x * (x - 1))
     [~1, 0, 1]                            Wert = [2, 0, 0]

(* c *)
let fun p 0 = true
      | p x = not (p (x - 1))
    fun q 1 = true
      | q x = not (q (x - 1))
in
  q 113
end                                        Wert = true
```

1.2 Auswertungsbäume

a) Geben Sie einen vollständigen Auswertungsbaum für den folgenden Ausdruck an. Die Auswertung findet in der leeren Umgebung statt. Siehe Anhang A für einige nützliche Auswertungsregeln.

```
if 4 < 2 then 3 else if 2 < 3 then ~1 else 1
```

Lösung

```
                            ------------  ------------
                            [] |= 2 => 2  [] |= 3 => 3
------------  ------------  --------------------------   --------------
[] |= 4 => 4  [] |= 2 => 2      [] |= 2 < 3 => true      [] |= ~1 => ~1
------------------------  ----------------------------------------------
   [] |= 4 < 2 => false          [] |= if 2 < 3 then ~1 else 1 => ~1
-------------------------------------------------------------------------
    [] |= if 4 < 2 then 3 else if 2 < 3 then ~1 else 1 => ~1
```

b) Geben Sie eine Umgebung an, die durch die Auswertung der folgenden Deklarationen entsteht. Die Ausgangsumgebung sei leer. Siehe Anhang A für einige nützliche Auswertungsregeln.

```
fun f (x : int) = x - 1
val y = 3
fun g (z : int) = f z < 2
```

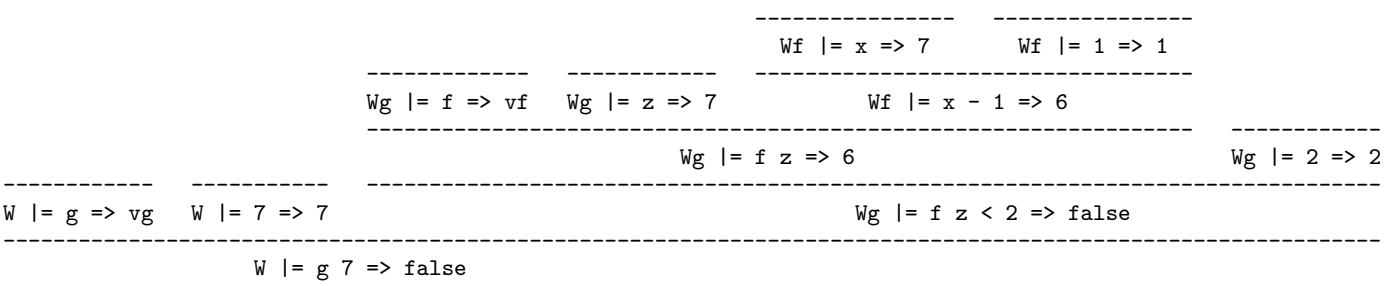
Lösung

```
f := (fun f x = x - 1, int -> int, [])
     ---------------------------------
                      |
                      vf
y := 3
g := (fun g z = f z < 2, int -> bool, [ f := vf ])
     ---------------------------------------------
                       |
                       vg
```

c) Sei W die Umgebung aus der vorherigen Frage. Geben Sie einen vollständigen Baum für die Auswertung des Ausdrucks `g 7` in der Umgebung W an. Siehe Anhang A für einige nützliche Auswertungsregeln.

Lösung

```
Wg := [f:=vf; g := vg; z:= 7]
Wf := [f:=vf; x:= 7]
```



Aufgabe 2 [20 Punkte] **Programmierung 1**

2.1 Kartesische und kaskadierte Prozeduren

a) Geben Sie die kartesische Version `foldl_c` der Prozedur `foldl` aus Anhang A an. Nennen Sie außerdem den Typ von `foldl_c`.

Lösung

```
foldl_c :: ('a * 'b -> 'b) * 'b * 'a list -> 'b

fun foldl_c (f, s, l) = foldl f s l
```

b) Geben Sie eine kaskadierte, end-rekursive Prozedur `fac` an, die die Fakultätsfunktion implementiert.

Lösung

```
(* mögliche Antwort 1 *)

fun fac1 acc 0 = acc
  | fac1 acc n = fac1 (acc * n)  (n - 1)


(* mögliche Antwort 2 *)

fun fac2 n =
  let fun fac' acc 0 = acc
| fac' acc n = fac' (acc * n) (n - 1)
  in fac' 1 n end

(* mögliche Antwort 3 *)

fun iter n s f = if n < 1 then s else iter (n - 1) (f s) f
fun fac3 n = #1(iter n (1, n) (fn (s, n) => (s * n, n - 1)))
```

2.2 Referenzen

Implementieren Sie die Prozedur `foldl` aus Anhang A mit Hilfe der Prozedur `map` (siehe Anhang A) und einer Referenz. Dabei soll die Referenz außerhalb von `foldl` unsichtbar sein.

Lösung

```
fun my_foldl f s l =
  let
    val acc = ref s
    val _ = map (fn x => acc := f (x, !acc)) l
  in !acc end
```

2.3 Matrizenrechnung

Wir stellen eine Zeile einer Matrix als eine Liste von Integern dar und implementieren eine Matrix als eine Liste von Zeilen. Zum Beispiel, wird die Matrix $\begin{pmatrix} 1 & 2 & 3 \\ 10 & 20 & 30 \end{pmatrix}$ durch `[[1,2,3], [10, 20, 30]]` dargestellt.

a) Geben Sie eine Prozedur `myrow : int -> int list -> bool` an, sodass `myrow n xs` den Wert `true` liefert, wenn das `n`-te Element von `xs` gleich `1` ist und alle anderen Elemente von `xs` gleich `0` sind. Das erste Element einer Zeile ist auf der Position 1. Zum Beispiel, liefert `myrow 2 [0, 1, 0, 0]` das Ergebnis `true`.

Lösung

```
fun myrow n l =
  let fun loop i n nil = true
        | loop i n (x :: xs) =
          (i = n andalso x = 1) orelse
          (i <> n andalso x = 0) andalso
          loop (i + 1) n xs
  in loop 1 n l end
```

b) Eine Matrix ist quadratisch falls die Zeilenanzahl gleich der Spaltenanzahl ist. Zum Beispiel, stellt `[[1,2], [10, 20]]` eine quadratische Matrix dar.

Geben Sie eine Prozedur `mysquare : int list list -> bool` an, sodass `mysquare xxs` das Ergebnis `true` liefert, wenn `xxs` eine quadratische Matrix darstellt.

Lösung

```
fun mysquare xss =
  let val len = length xss in List.all (fn xs => length xs = len) xss end
```

c) Die Einheitsmatrix ist eine quadratische Matrix, bei der jedes Element auf der Diagonale gleich 1 und alle anderen Elemente gleich 0 sind. Zum Beispiel, stellt `[[1,0,0], [0, 1, 0], [0,0,1]]` eine Einheitsmatrix dar.

Geben Sie eine Prozedur `myunit : int list list -> bool` an, sodass `myunit xxs` das Ergebnis `true` liefet, wenn `xxs` eine Einheitsmatrix darstellt. Benutzen Sie dabei `myrow`.

Lösung

```
(* mögliche Antwort 1 *)

fun myunit1 xss =
  mysquare xss andalso
  #1(foldl (fn (xs, (s, i)) => (s andalso myrow i xs, i + 1)) (true, 1) xss)


(* mögliche Antwort 2 *)

fun myunit2 xss =
  let fun diag n nil = true
        | diag n (xs :: xss) = myrow n xs andalso diag (n + 1) xss
  in mysquare xss andalso diag 1 xss end
```

Aufgabe 3 [15 Punkte] **Programmierung 2**

Wir modellieren Getränke mit Hilfe des Typs `drink`, der wie folgt auf den Typen `liquid` und `amount` aufbaut.

```
datatype liquid = Water | Juice | Coke
type amount = int
datatype drink = Basic of liquid * amount | Mix of drink list
```

`Mix [Basic (Water, 10), Basic (Juice, 10)]` stellt zum Beispiel ein Getränk dar, das aus zehn Wasser- und zehn Safteinheiten zubereitet wurde.

a) Geben Sie eine Prozedur `unit : liquid -> int` an, die die Kalorienanzahl in einer Flüssigkeitseinheit wie folgt bestimmt. Eine Wassereinheit hat keine Kalorien, eine Safteinheit hat eine Kalorie und eine Coke-Einheit hat zwei Kalorien.

Lösung

```
fun unit Water = 0
  | unit Juice = 1
  | unit Coke = 2
```

b) Geben Sie eine Prozedur `total : drink -> int` an, die die Gesamtzahl der Kalorien in einem Getränk berechnet. Zum Beispiel, liefert `total (Mix [Basic (Water, 10), Basic (Juice, 10)])` das Ergebnis `10`.

Lösung

```
fun total (Basic (l, a)) = a * unit l
  | total (Mix ds) = foldl (fn (d, t) => t + total d) 0 ds
```

c) Geben Sie eine Prozedur `ingredients : drink -> (amount * amount * amount)` an, die Wasser-, Saft- und Coke-Gehalt eines Getränks berechntet. Zum Beispiel, liefert `ingredients (Mix [Basic (Water, 10), Basic (Juice, 10)])` das Ergebnis `(10, 10, 0)`.

Lösung

```
fun ingredients d =
  let fun ingredients_ (d, (w, j, c)) =
    case d of
      Basic (l, a) =>
        (case l of
           Water => (w + a, j, c)
         | Juice => (w, j + a, c)
         | Coke => (w, j, c + a))
    | Mix ds => foldl ingredients_ (w, j, c) ds
  in ingredients_ (d, (0, 0, 0)) end
```

Aufgabe 4 [15 Punkte] **Programmierung 3**

Ein gerichteter Graph besteht aus einer Menge von Kanten. Eine Kante ist ein Paar von Knoten, wobei jeder Knoten durch einen Integer dargestellt wird. Wir stellen einen Graphen als eine Liste von Integer-Paaren dar, z.B., `[(1, 2), (2, 3), (1, 3), (4, 3)]`.

a) Geben Sie eine Prozedur `next : int -> (int * int list) -> (int list)` an, sodass `next n edges` eine Liste der Knoten liefert, die mit dem Knoten `n` durch eine aus `n` ausgehende Kante verbunden sind. Zum Beispiel, liefert `next 1 [(1, 2), (2, 3), (1, 3), (4, 3)]` das Ergebnis `[2, 3]`, weil die Kanten `(1,2)` und `(1, 3)` in der Eingabeliste vorkommen. (Mehrfaches Auftreten von Knoten in der Ergebnisliste ist zulässig.)

Lösung

```
fun next n edges =
  let
    val relevant_edges = List.filter (fn (a, _) => a = n) edges
    val successors = map #2 relevant_edges
  in successors end
```

b) In dieser Frage betrachten wir nur azyklische Graphen, d.h., Graphen ohne zyklische Kantenfolgen. Geben Sie eine Prozedur `reach : int -> (int * int list) -> (int list)` an, sodass `reach n edges` eine Liste von Knoten liefert, die aus dem Knoten `n` über eine Kantenfolge erreichbar sind. Diese Folge kann auch leer sein. Zum Beispiel, liefert `reach 1 [(1, 2), (2, 3), (1, 3), (4, 3)]` das Ergebnis `[1, 2, 3]` und liefert `reach 3 [(1, 2), (2, 3), (1, 3), (4, 3)]` das Ergebnis `[3]`. (Mehrfaches Auftreten von Knoten in der Ergebnissliste ist zulässig.)

Lösung

```
(* mögliche Antwort 1 *)

fun reach1 n edges =
  foldl (fn ((a, b), reached) =>
         if a = n then reach1 b edges @ reached else reached) [n] edges

(* mögliche Antwort 2 *)

fun reach2 n edges =
  let val succs = next n edges
  in foldl (fn (succ, reached) => reach2 succ edges @ reached) [n] succs end
```

Aufgabe 5 [15 Punkte] **Programme als diskrete Strukturen**

Wir betrachten eine mathematische Prozedur `upto : int * int -> unit`, die wie folgt definiert wird.

```
upto (i, n) = if i < n then upto (i+1, n) else ()
```

a) Geben Sie den Definitionsbereich und eine natürliche Terminierungsfunktion für `upto` an.

Lösung

```
Definitionsbereich: Z x Z
Terminierungsfunktion: \lambda (i, n) \in Z x Z . | n - i |
```

b) Geben Sie die Rekursionsfunktion und die Rekursionsrelation für `upto` an.

Lösung

```
Rekursionsfunktion: \lambda (i, n) \in Z x Z . if i > n then <(i + 1, n)> else <>
Rekursionsrelation: R = { ((i,n), (i+1, n)) | (i,n) \in Z x Z ^ i < n }
```

c) Geben Sie die Ergebnisfunktion von `upto` an. Beweisen Sie die Korrektheit Ihrer Antwort.

Lösung

```
Ergebnisfunktion: f := \lambda (i,n) \in Z x Z . ()
Korrektheit:
We have to check that
1) Dom f \subseteq Dom p, and that
2) f satisfies the defining equations of upto.
Let n be given. We proceed by case distinction over i:
Case i = n: Prove that f (i, n) = (). The equation holds by the defn. of f.
Case i < n: Prove that f (i, n) = f (i + 1, n).
            f (i, n) = ()
                     = f (i + 1, n)
```

Aufgabe 6 [20 Punkte] **Programmverifikation durch induktive Beweise**

Wir betrachten eine mathematische Prozedur `foldl : (X * Y -> Y) * Y * Lists(X) -> Y`, die wie folgt definiert wird.

```
foldl (f, s, nil) = s
foldl (f, s, x::xr) = foldl (f, f(x, s), xr)
```

Dazu betrachten wir eine mathematische Funktion `f` $\in$ `int * int -> int`, sodas
`f (x, s) = s + (if x >= 0 then x else -x)`.

Beweisen Sie, dass für jeden Integer `s` und jede Integer-Liste `xs` das Ergebnis der Anwendung `foldl (f, s,xs)` größer oder gleich `s` ist.

Lösung

```
 Prove that \forall xs, s . foldl (f, s, xs) >= s.

 We prove that \forall s . foldl (f, s, xs) >= s by induction over xs.

 1) Case xs = nil
 Let s be given. Then foldl (f, s, nil) = s >= s

 2) Case xs = x :: xs'
 Let the induction hypothesis be \forall s . foldl (f, s, xs') >= s.
 Let s be given. Then
 foldl (f, s, xs) =  foldl (f, s, x :: xs')
                  =  foldl (f, f (x, s), xs')          defn. foldl
                  >= f (x, s)                          I.H.
                  >= s + (if x >= 0 then x else ~x)    defn. f
                  >= s + |x|                           Prop. A
                  >= s

------------------------------

Proposition A: \forall x . (if x >= 0 then x else ~x) = |x|
Proof: Case distinction over x.
 Case x >= 0
   (if x >= 0 then x else ~x) = x
                              = |x|
 Case x < 0
   (if x >= 0 then x else ~x) = ~x
                              = |x|
```

A Anhang

$$\frac{V(b) = v}{V \models b \Rightarrow v} \qquad \frac{}{V \models k \Rightarrow k} \qquad \frac{V \models e_1 \Rightarrow v_1 \quad V \models e_2 \Rightarrow v_2}{V \models e_1 \circ e_2 \Rightarrow v_1 \circ v_2} \qquad \frac{V \models e_1 \Rightarrow \mathit{true} \quad V \models e_2 \Rightarrow v}{V \models \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \Rightarrow v}$$

$$\frac{V \models e_1 \Rightarrow \mathit{false} \quad V \models e_3 \Rightarrow v}{V \models \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \Rightarrow v}$$

$$\frac{V \models e_1 \Rightarrow (\text{fun } f\ b = e,\ t,\ V_1) \quad V \models e_2 \Rightarrow v_2 \quad V_1 + [f := (\text{fun } f\ b = e,\ t,\ V_1)] + [b := v_2] \models e \Rightarrow v}{V \models e_1\ e_2 \Rightarrow v}$$

$$\frac{V_1 = (V \text{ eingeschränkt auf } \mathit{FreeIds}(\text{fun } f\ (b\ :\ t_1)\ :\ t_2 = e))}{V \mid\gg \big(\text{fun } f\ (b\ :\ t_1)\ :\ t_2 = e\big) : V + [f := (\text{fun } f\ b = e,\ t_1 \to t_2,\ V_1)]}$$

$$\frac{V_0 \mid\gg d_1 : V_1 \quad \ldots \quad V_{n-1} \mid\gg d_n : V_n}{V_0 \mid\gg d_1\ \ldots\ d_n : V_n} \qquad \frac{V \models e \Rightarrow v}{V \mid\gg \text{val } b = e : V + [b := v]}$$

```
fun map f nil     = nil
  | map f (x::xr) = (f x) :: (map f xr)

map : ('a -> 'b) -> 'a list -> 'b list

fun filter f nil     = nil
  | filter f (x::xr) = if f x then x :: filter f xr
  else filter f xr

filter : ('a -> bool) -> 'a list -> 'a list

fun exists f nil     = false
  | exists f (x::xr) = f x orelse exists f xr

exists : ('a -> bool) -> 'a list -> bool

fun all f nil     = true
  | all f (x::xr) = f x andalso all f xr

all : ('a -> bool) -> 'a list -> bool

fun foldl f s nil     = s
  | foldl f s (x::xr) = foldl f (f(x,s)) xr

foldl : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b

fun foldr f s nil     = s
  | foldr f s (x::xr) = f(x, foldr f s xr)

foldr : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b

fun length nil     = 0
  | length (x::xr) = 1 + length xr

length : 'a list -> int

explode : string -> char list

implode : char list -> string
```