TENTAMEN I DVA 229 FUNKTIONELL PROGRAMMERING MED F#

Torsdagen den 23 mars 2023, kl 14:30 – 18:30

LÖSNINGSFÖRSLAG

UPPGIFT 1 (6 POÄNG)

a) En enkel rekursion genom listan, där vi räknar antalet förekomster av "`None`":

```
let rec countNone l =
  match l with
  | [] -> 0
  | None :: xs -> 1 + countNone xs
  | Some x :: xs -> countNone xs
```

b) En variant där vi först gör om listan till att bestå av nollor och ettor och sen summerar listan:

```
let  countNone l =
  l |> List.map (fun x -> if x = None then 1 else 0) |> List.fold (+) 0
```

UPPGIFT 2 (8 POÄNG)

a) Den lokala funktionen "`search`" rekurserar genom arrayens index på jakt efter det största elementet:

```
let largest (a : float []) =
  let rec search i n acc =
    if i = n
    then acc
    else if a.[i] > acc
         then search (i+1) n (a.[i])
         else search (i+1) n acc
  in search 0 (Array.length a) (a.[0])
```

b) Lösningen blir mycket snarlik dden förra. Där sparas det hittills största värdet i ett funktionsargument, medan här den sparas i referenscellen som uppdateras vid varje nytt anrop till den inre funktionen:

```
let largest (a : float []) =
  let acc = ref (a.[0])
  let rec search i n =
    if i = n
    then !acc
    else
      if a.[i] > !acc
      then acc := a.[i]
      else ()
      search (i+1) n
  in search 0 (Array.length a)
```

OBS att p.g.a. värderestriktionen är typningen av arrayen `a` nödvändig för att typinferensen ska fungera. Jag har dock inte dragit några poäng om typningen fattas.

UPPGIFT 3 (3 POÄNG)

a) `'a -> unit`

b) I det första uttrycket får `f` en (lång) lista som argument. Eftersom F# använder call-by-value för listor måste hela listan räknas ut innan anropet kan ske. I det andra uttrycket får `f` i stället en sekvens. Dessa räknas ut on-demand (lat evaluering). Eftersom `f` överhuvudtaget inte använder sitt argument kommer anropet att terminera utan att sekvensen räknas ut alls.

Alltså bör det andra uttrycket (med sekevnsen) gå fortast att räkna ut.

UPPGIFT 4 (2 POÄNG)

`x` får typen `float` och `y` får typen `int`. Detta leder till typfel i additionen, för man får inte addera numeriska värden av olika typ i F#.

UPPGIFT 5 (1 POÄNG)

Deklarationerna definierar exakt samma funktion.

UPPGIFT 6 (6 POÄNG)

a)

```
type Tritree<'a> = Leaf of 'a | Single of 'a * Tritree<'a>
                   | Double of 'a * Tritree<'a> * Tritree<'a>
                   | Triple of 'a * Tritree<'a> * Tritree<'a> * Tritree<'a>
```

b) En enkel rekursion genom trädet, där vi matchar ut delträden i de olika möjliga fallen:

```
let rec count_leaves t =
  match t with
  | Leaf _       -> 1
  | Single (_,t1)    -> count_leaves t1
  | Double (_,t1,t2) -> count_leaves t1 + count_leaves t2
  | Triple (_,t1,t2,t3) -> count_leaves t1 + count_leaves t2 + count_leaves t3
```

UPPGIFT 7 (4 POÄNG)

Vi vet:

```
0, 1 : int
(-) : 'n -> 'n -> 'n, 'n numerisk typ
_ : 'a
```

Vi antar nu, för variablerna i deklarationen:

```
f : 'b
x : 'c
```

Vi kollar nu att alla uttryck är typkorrekta samt att typen på vänsterledet (VL) är lika med typen på högerledet (HL). Under den processen kommer villkoren på typen för `f` successivt att skärpas. När processen är klar kommer dessa villkor att ge den mest generella typen för `f`.

Först kollar vi VL. Där appliceras `f` på `x`. Detta är typkorrekt endast om `f` är funktionstypad med typen för `x` som argumenttyp, dvs.

```
'b = 'c -> 'd
```

för någon typvariabel `'d`. Vidare har VL typen `'d`. Låt oss nu gå igenom högerledet (HL), som är ett match-uttryck. Först kollar vi att alla mönster har samma typ. dvs. typerna för `0` och _ är lika. Detta är fallet om `'a = int`. Vidare måste mönstren och uttrycket som matchas (`x`) ha samma typ, vilket är fallet om `'c = int`. Så nu vet vi följande:

```
x : int
f : int -> 'd
```

Vidare måste de uttryck som returneras (`x`, `f(x-1)`) ha samma typ. Eftersom `x : int` så måste `f(x-1) : int`, vilket är fallet om

```
'd = int
```

Så nu har vi

```
f : int -> int
x : int
```

Vi måste också kolla att `f(x-1)` är váltypat. `x-1` måste då ha samma typ som argumenttypen för `f` (`int`), vilket är fallet om `'n = int`. Då måste också argumenten till "`-`" (`x`, `1`) ha typ `int`, vilket stämmer.

Återstår att kolla att VL och HL har samma typ. VL har typ `int`. Typen för HL är lika med typen för hela matchuttrycket som är samma som den för de returnerade uttrycken, dvs. `int`, så typerna för båda leden är `int`. Således får vi följande typning för `f`:

```
f : int -> int
```

Att detta är den mest generella typen för `f` följer av att vi i varje steg gjort minimala antaganden om typningen på de olika deluttrycken.