TENTAMEN I DVA 201 FUNKTIONELL PROGRAMMERING MED F#

Torsdagen den 7 januari 2016, kl 8:10 – 12:30

LÖSNINGSFÖRSLAG

UPPGIFT 1 (4 POÄNG)

Vi gör en lösning med två ömsesidigt rekursiva funktioner: en som lägger det första elementet från den första listan först och en som lägger det första elementet från den andra listan listan först. Dessa får sen omväxlande anropa varandra. När en lista tar slut fyller båda funktionerna på med resten av den andra listan:

```
let rec interleave l1 l2 =
  match (l1,l2) with
  | (x::xs,ys)  -> x::interleave2 xs ys
  | ([],ys) -> ys
and interleave2 l1 l2 =
  match (l1,l2) with
  | (xs,y::ys)  -> y::interleave xs ys
  | (xs,[]) -> xs
```

UPPGIFT 2 (2 POÄNG)

I deklarationen av `f` kommer typen för `(+)` att defaulta till `int -> int -> int`. Detta gör att både argumentet `x` liksom funktionskroppen `x + x` får typen `int`, vilket i sin tur medför att `f` får typen `int -> int`.

I deklarationen av `g` typas `x` till `float`. Detta medför att typen för `(+)` blir `float -> float -> float` samt att `g` får typen `float -> float`.

UPPGIFT 3 (4 POÄNG)

Vi deklarerar en funktion som rekurserar sig genom talföljden. Rekursionen sker i en lokalt definierad funktion. Den funktionen har också ett ackumulerande argument som räknar antalet rekursiva anrop och därmed också längden på talföljden:

```
let even n = n % 2 = 0
let sequence n =
 let rec seq_local n count =
    match n with
    | 1             -> count
    | _ when even n -> seq_local (n/2) (count + 1)
    | _             -> seq_local (3*n + 1) (count + 1)
  in seq_local n 1
```

UPPGIFT 4 (2 POÄNG)

Lat evaluering: evalueringen av ett funktionsargument räknas inte ut förrän dess värde verkligen behövs. **Ivrig evaluering**: funktionsargument räknas ut helt innan anropet till funktionen sker. F# har ivrig evaluering.

UPPGIFT 5 (4 POÄNG)

Den här uppgiften går att göra på några olika sätt. Ett sätt är att först mappa en funktion över arrayen som tar `None` till `0.0` och `Some x` till `x` och sen summera alla elementen med `Array.sum`:

```
let opt2float v =
  match v with
  | None   -> 0.0
  | Some x -> x

let add_opt_array = Array.map opt2float >> Array.sum
```

UPPGIFT 6 (4 POÄNG)

En lösning som använder rekursion tillsammans med sekvensering för att få utskrifterna att komma i rätt ordning. En lokal rekursiv funktion, med ett extra, ackumulerande argument för radnumret, gör själva jobbet. Vi använder `printf` med en lämplig formatsträng för utskrifterna:

```
let printlist l =
  let rec print_local l line =
    match l with
    | x::xs -> printf "%d: %A\n" line x; print_local xs (line + 1)
    | []    -> ()
  in print_local l 1
```

UPPGIFT 7 (6 POÄNG)

a)

```
type BTree<'a> = Leaf of 'a | One of ('a * BTree<'a>)
               | Two of ('a * BTree<'a> * BTree<'a>)
```

b) En enkel rekursion genom trädet där vi formar listan i den rätta "inorder"-ordningen:

```
let rec inorder t =
    match t with
    | Leaf x        -> [x]
    | One (x,t)     -> x :: inorder t
    | Two (x,t1,t2) -> x :: (inorder t1 @ inorder t2)
```

UPPGIFT 8 (4 POÄNG)

Ett annat sätt att uttrycka likheten är att

$$\texttt{map id xs} = \texttt{xs}$$

för alla listor `xs`. (För att förenkla skriver vi `map` istället för `List.map`.) Vi visar nu detta med induktion över listor. Vi väljer direkt

$$P(\texttt{xs}) \iff \texttt{map id xs} = \texttt{xs}$$

Induktionsbeviset kommer då att ge $\forall \texttt{xs}.P(\texttt{xs})$ vilket är precis det resultat vi vill ha.

Vi får använda funktionsdeklarationerna av `map` och `id` i beviset. De ger upphov till följande ekvationer:

$$\texttt{map f}\ [\,] = [\,] \tag{1}$$

$$\texttt{map f}\ (\texttt{x} :: \texttt{xs}) = \texttt{f x} :: (\texttt{map f xs}) \tag{2}$$

$$\texttt{id x} = \texttt{x} \tag{3}$$

Basfall, visa att $P([\,])$ gäller, dvs. `map id` $[\,] = [\,]$. Detta följer direkt från (1) (med `f` $=$ `id`).

Induktionssteg: Antag att $P(\texttt{xs})$ ("induktionshypotesen") gäller, visa att $P(\texttt{x::xs})$ gäller för godtyckligt `x`. $P(\texttt{xs})$ är samma som

$$\texttt{map id xs} = \texttt{xs} \tag{4}$$

Vi vill visa $P(\texttt{x} :: \texttt{xs})$, dvs.

$$\texttt{map id}\ (\texttt{x} :: \texttt{xs}) = \texttt{x} :: \texttt{xs}$$

Låt oss visa att vänsterledet (VL) är lika med högerledet (HL). Vi har, för godtyckligt `x`, att

$$\begin{aligned}
\text{VL} &= \texttt{map id}\ (\texttt{x} :: \texttt{xs}) \\
&= \text{(pga. (2))} \\
&= \texttt{id x} :: \texttt{map id xs} \\
&= \text{(pga. (3))} \\
&= \texttt{x} :: \texttt{map id xs} \\
&= \text{(pga. induktionshypotesen (4))} \\
&= \texttt{x} :: \texttt{xs} \\
&= \text{HL}
\end{aligned}$$

Detta visar induktionssteget.

Eftersom vi visat både basfall och induktionssteg har vi visat $\forall \texttt{xs}.P(\texttt{xs})$, vilket ger det sökta resultatet.