Norsk

# Universitetet i Bergen
## Det matematisk-naturvitenskapelige fakultet

| | | |
|---|---|---|
| Eksamen i | : | INF-122 Funksjonell programmering |
| Dato | : | 8 desember 2017 |
| Tid | : | 9:00 – 12:00 |
| Antall sider | : | 3 |
| Tillatte hjelpemidler | : | Ingen |

- Prosentsatsene angir *kun omtrentlig* vekting ved sensur og forventet tidsforbruk
- Løsninger av delproblemer som du ikke har besvart kan antas gitt dersom de trenges i andre delproblemer.
- Angi typen til enhver funksjon du programmerer.
- Programmer og forklar alle hjelpefunksjoner som du selv innfører. Din kode skal ikke forutsette andre funksjoner enn de som er tilgjengelige fra standard `Prelude`.

## 1 Programmer følgende funksjoner: (25%)

**1.1.** `harEl::(t->Bool)->[t]->Bool`, slik at `harEl pr xs = True` hvis listen `xs` har et element `x` som tilfredstiller predikatet `pr`, dvs., slik at `pr x = True`, og `False` ellers. F.eks.:

```
harEl (==3) [1,1,2,3,2] = True
harEl (<3) [1,1,2,3,2] = True
harEl (>5) [1,1,2,3,2] = False.
```

**1.2.** `el::(t->Bool)->[t]->t`, slik at `el pr xs` returnerer første elementet x fra listen xs som tilfredstiller predikatet pr. Funksjonen antar at et slikt element finnes i listen. F.eks.:

```
el ((=='a').fst) [('b',2),('a',3),('a',4)] = ('a',3)
el ((>3).snd) [('b',2),('a',3),('a',4)] = ('a',4).
```

**1.3.** `gRep::(t->Bool)->t->[t]->[t]`, slik at `gRep pr y xs` erstatter med y, ethvert element x fra listen xs som tilfredstiller predikatet `pr`. F.eks.:

```
gRep (<'d') 'z' ``abcd'' = ``zzzd''
gRep (=='a') 'x' ``abcbcac'' = ``xbcbcxc''.
```

**1.4.** Vi bruker binære trær med heltall lagret i alle noder (inklusivt blader) definert ved: `data BT = B Int | N BT Int BT`. Programmer følgende funksjoner:
**(a)** `elt::BT->Int->Bool`, slik at `elt tr x = True` hvis tallet `x` forekommer i treet `tr`, og `False` ellers, f.eks., `elt (N (B 1) 2 (B 0)) 2 = True` og `elt (B 1) 2 = False`.
**(b)** `toL::BT->[Int]`, slik at `toL tr` er en liste med alle tall som forekommer i treet `tr`, f.eks., `toL (N (B 1) 2 (N (B 3) 3 (B 0))) = [1,2,3,3,0]`.
**(c)** `dup::BT->Bool`, slik at `dup tr = True` hvis noen tall forekommer (minst) to ganger i treet `tr`, og `False` ellers. For eksempel, `dup (N (B 1) 2 (N (B 3) 5 (B 0))) = False` og `dup (N (B 1) 2 (B 2)) = True`.

## 2 Rettede grafer **(40%)**

En (rettet) graf er en mengde av noder med kanter som forbinder utvalgte par av noder (i én retning, fra kilde- til målnode). F.eks.: følgende graf `gr` har 5 noder og 6 kanter:

$$\begin{tikzcd} a \arrow[r] \arrow[d] & b \arrow[d] \\ c \arrow[ur] & d \arrow[l] \arrow[r] & e \end{tikzcd}$$

`gr`

| | |
|---|---|
| noder: | $[a, b, c, d, e]$ |
| kanter, KL = | $[(a, b), (a, c), (b, d), (c, b), (d, c), (d, e)]$ |

En graf kan representeres som en *kantliste*, nemlig en liste hvis elementer er alle par av noder tilsvarende kanter. F.eks., representerer kantlisten KL grafen `gr`. (Vi antar at hver node er med i minst én kant, slik at vi ikke representerer noder i tillegg til kanter.) Alternativt, kan grafen representeres som en *naboliste*, nemlig, en liste av par der første elementet er en node, si $x$, og andre er listen av noder med kanter fra denne noden $x$. F.eks., er grafen `gr` representert av nabolisten NL $= [(a, [b, c]), (b, [d]), (c, [b]), (d, [c, e]), (e, [])]$.

**2.1.** Programmer en funksjon `naboL::Eq t => [(t,t)]->[(t,[t])]` som konverterer kantliste representasjon av en graf til dens naboliste, f.eks., `naboL` KL kan gi NL.

**2.2.** Programmer en invers funksjon `kantL::[(t,[t])]->[(t,t)]` som konverterer naboliste representasjon av en graf til dens kantliste, f.eks., `kantL` NL kan gi KL.

**2.3.** En sti er en liste med noder $[x_1, x_2, ..., x_n]$ slik at hvert par $(x_1, x_2), (x_2, x_3), ..., (x_{n-1}, x_n)$ er en kant. En syklus er en sti som starter og slutter i samme node, dvs. der $x_1 = x_n$. (Man følger kanter kun fra kilde til mål, dvs., kun i pilens retning.)

For resten av denne oppgaven, velg enten en kant- eller en nabolisterepresentasjon. Si hva du har valgt. Valget ditt betegner vi med `??`.

**(a)** Programmer en funksjon `naboer::??->t->[t]`, slik at `naboer g x` returnerer listen av alle naboer til noden `x` (alle noder som har en kant fra `x`) i grafen `g`. I grafen `gr` over, `naboer gr a = [b,c]` og `naboer gr e = []`.

**(b)** Programmer nå en funksjon `cyc::??->t->[t]`, slik at `cyc g x` returnerer stien tilsvarende en syklus når en slik kan nås ved å starte fra noden `x` i grafen `g`, og tom sti [] ellers. F.eks., for grafen `gr` over,

`cyc gr e = []` – siden ingen syklus kan nås fra $e$,

`cyc gr b = [b,d,c,b]`,

`cyc gr a = [a,b,d,c,b]` eller `cyc gr a = [a,c,b,d,c]`.

I det siste tilfellet er det opp til deg å bestemme om funksjonen returnerer hele stien fom. noden tom. syklusen, eller bare syklusen (dvs. bare `[b,d,c,b]` eller `[c,b,d,c]`.)

[[Hint: Det er hovedsaklig korrekthet, og ikke effektivitet, som teller her. Det kan være lurt å programmere en hjelpefunksjon `trav::??->t->...`, slik at `trav gr x...` besøker alle stier utgående fra node `x` i grafen `gr`, uten å gå i en syklus. Den kan ha bruk for et ekstra argument som samler alle noder besøkt på den aktuelle stien.]]

## 3 Litt IO **(15%)**

Programmer en funksjon (samt alle hjelpefunksjoner) `main::IO ()` som gir brukeren mulighet til å utføre følgende kommandoer:

– g : oppretter en ny, tom graf

– k x y : legger kanten (x,y) til grafen (også hvis node x eller y ikke finnes fra før)

– f x y : fjerner kanten (x,y) fra grafen
– s : viser en vilkårlig syklus i den aktuelle grafen eller tom sti, hvis grafen er asyklisk
– q : avslutter programmet.

Dersom det har noen betydning, spesifiser hvilken grafrepresentasjon du bruker.

## 4 Typeinferens **(20%)**

Vi betrakter to følgende uttrykk:

(a) `\h -> \x -> (h x) h` og (b) `\h -> \x -> (h x) x`.

Et av dem har en type i Haskell, mens det andre har det ikke. Velg det uttrykket som ***ikke*** har en type, vis hele typeavledning (så langt den går) og grunnen til at den feiler.

(Hvis du har tid, vis også avledning av typen for det andre uttrykket.)

Lykke til!
Michał Walicki

# ..................................Vedlegg..................................

## Hindley-Milner typeinferens: transformasjonsalgoritme

| | input | ⇒ | output |
|---|---|---|---|
| (t1) | $E(\Gamma \mid con :: t)$ | $\Rightarrow$ | $\{t = \theta(con)\}$ |
| | | | – typen til en konstant slås opp i ordboken |
| (t2) | $E(\Gamma \mid x :: t)$ | $\Rightarrow$ | $\{t = \Gamma(x)\}$ |
| | | | – typen til en variabel sjekkes i konteksten |
| (t3) | $E(\Gamma \mid f\ g :: t)$ | $\Rightarrow$ | $E(\Gamma \mid g :: a) \cup E(\Gamma \mid f :: a \to t)$ |
| | | | – $a$ er en *fersk* typevariabel |
| (t4) | $E(\Gamma \mid \backslash x \to ex :: t)$ | $\Rightarrow$ | $\{t = a \to b\} \cup E(\Gamma, x :: a \mid ex :: b)$ |
| | | | – $a, b$ er *ferske* typevariabler |

## Martelli-Montanari unifikasjonsalgoritme

| | input | ⇒ | output | forutsatt at : |
|---|---|---|---|---|
| $(u1)$ | $E, t = t$ | $\Rightarrow$ | $E$ | |
| $(u2)$ | $E, f(t_1...t_n) = f(s_1...s_n)$ | $\Rightarrow$ | $E, t_1 = s_1, ..., t_n = s_n$ | |
| $(u3)$ | $E, f(t_1...t_n) = g(s_1...s_m)$ | $\Rightarrow$ | $NO$ | $f \neq g$ eller $n \neq m$ |
| $(u4)$ | $E, f(t_1...t_n) = x$ | $\Rightarrow$ | $E, x = f(t_1...t_n)$ | |
| $(u5)$ | $E, x = t$ | $\Rightarrow$ | $E[x/t], x = t$ | $x \notin Var(t)$ |
| $(u6)$ | $E, x = t$ | $\Rightarrow$ | $NO$ | $x \in Var(t)$ |

## Oppgave 1 – løsningsforslag (25%)

**1.1.** `harEl pr = any pr`, eller `harEl pr xs = not(null (filter pr xs))`, eller:

```
harEl pr [] = False
harEl pr (y:ys) = if (pr y) then True else harEl pr ys
```

**1.2.** `el pr (z:zs) = if (pr z) then z else el pr zs`

**1.3.** erstatt alle forekomster av elementer fra listen som tilfredstiller test med y

```
gRep pr y = map (\x -> if (pr x) then y else x)
```

Noen kan hende bommer på det generiske, så får noen få poeng for plain:

```
rep x y ls = map (\z -> if x==z then y else z) ls
```

**1.4.** data BT = B Int | N BT Int BT

```
elt (B x) y = x==y
elt (N l v r) y = v==y || elt l y || elt r y
toL (B x) = [x] toL (N l v r) = (toL l) ++ [v] ++ (toL r)
dup tr = dupL (toL tr)
dupL [] = False
dupL (x:xs) = elem x xs || dupL xs
```

## Oppgave 2 – løsningsforslag (40%)

**2.1.** naboL :: (Eq a, Eq t) => [(t, a)] -> [(t, [a])]

```
naboL xs = foldr addk [] xs
addk (a,b) nL = if (harEl ((== a).fst) nL) then
                      let (x,y) = el ((== a).fst) nL in gRep (==(x,y)) (x,b:y) nL
                   else (a,[b]):nL
```

**2.2.**

```
kantL ::  [(t,[t1])] -> [(t,t1)]
kantL xs = foldr (\(f,nl) ak -> [(f,n)|n<-nl]++ak) [] xs
```

**2.3.** Bruker naboliste, siden den gir raskere adgang til naboer av en gitt node.

**(a)** naboer nL x = if (harEl ((== x).fst) nL) then snd (el ((== x).fst) nL) else []

**(b)**

```
cyc nL x = let re = trav nL [] x in
             if null re then [] else reverse (head re)
trav nL vis x = if (elem x vis) then [x:vis]
                  else concat (map (trav nL (x:vs)) (naboer nL x))
```

## Oppgave 3 – løsningsforslag (15%)

```
main = exc []

exc :: [(t,t)] -> IO ()

exc gr = do
 putStrLn ”g / (k/f) x y / s / q”
 c <- getLine
 let com = words c
 let m = head com
 if (m == “q”) then return ()
 else if (m == “k” || m== “f”) then do
        let x = read (head (tail com)) :: Int
            y = read (head (tail (tail com))) :: Int
        if (m == “k”) then exc ((x,y):gr) – – bruker kantliste her og under
        else exc (filter (/=(x,y)) gr)
 else if (m == “g”) then exc []
 else if (m == “s”) then do print (cycG (naboL gr))
                             exc gr
 else do print “Ukjent kommando”
           exc gr
```

## Oppgave 4 – løsningsforslag (20%)

**(a)** `\h -> \x -> (h x) h` – typing feiler ved occurs check:

$$
\begin{array}{ll}
E(\emptyset \mid \backslash h \to \backslash x \to (h\ x)\ h :: t) = & \{t = a \to d\} \\
E(h :: a \mid \backslash x \to (h\ x)\ h :: d) = & \{t = a \to d\} \\
E(h :: a, x :: b \mid (h\ x)\ h :: c) = & \{d = b \to c, t = a \to d\} \\
E(h :: a, x :: b \mid h :: e) \cup & \\
\quad E(h :: a, x :: b \mid h\ x :: e \to c) = & \{d = b \to c, t = a \to d\} \\
E(h :: a, x :: b \mid h\ x :: e \to c) = & \{e = a, d = b \to c, t = a \to d\} \\
E(h :: a, x :: b \mid x :: f) \cup & \\
\quad E(h :: a, x :: b \mid h :: f \to e \to c) = & \{e = a, d = b \to c, t = a \to d\}
\end{array}
$$

$\{a = f \to e \to c, f = b, e = a, d = b \to c, t = a \to d\}$

$\{a = b \to a \to c, f = b, e = a, d = b \to c, t = a \to d\}$ – occurs check ved første ligning.

**(b)** `\h -> \x -> (h x) h ::` $(a \to a \to b) \to a \to b$.