Universitetet i Bergen
Det matematisk-naturvitenskapelige fakultet

Eksamen i	:	INF-122 Funksjonell programmering
Dato	:	21 Februar 2018
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.
- Programmer og angi typen til alle hjelpefunksjoner som du selv innfører. Din kode skal ikke forutsette andre moduler enn standard `Prelude`.

1 Programmer følgende funksjoner: (25%)

I denne oppgaven betrakter vi lister som en representasjon av mengder, dvs. samlinger av elementer der rekkefølgen og multiplisitet ikke spiller noen rolle. F.eks., representerer lister [1,2,3] og [3,2,2,3,1,2] den samme mengden $\{1, 2, 3\}$. Programmer følgende funksjoner – besvarelsen til **1.1.** og **1.2.** ***skal benytte mønstre***.

1.1. `mengde::EQ t => [t] -> Bool` gir `True` hvis alle elementer i argumentlisten er forskjellige, og `False` ellers. F.eks., `mengde [1,2,3] = True` og `mengde "abcb" = False`.

1.2. `rep:EQ t => [t]->[t]` returnerer mengden representert av inputlisten, dvs., en liste med alle elementene fra inputlisten, men uten noen duplikater. Det skal altså alltid gjelde at `mengde(rep xs) = True` (samt at for enhver `x`: `elem x xs = elem x (rep xs)`).

1.3. `del::EQ t => [t]->[t]->Bool` gir `True` hvis første argumentet er en delmengde av andre, dvs., hvert element som forekommer i den første listen, forekommer også i den andre. F.eks.: `del "cbbbc" "bca" = True` og `del [2,3] [3,3,1] = False`.

1.4. `eq::EQ t => [t]->[t]->Bool` gir `True` hvis to lister har de samme elementene og `False` ellers, f.eks.: `eq "cbac" "bbca" = True` og `eq [1,2,3] [3,1] = False`.

1.5. `eqG::(t->t->Bool)->[t]->[t]->Bool` er en parametrisert versjon av `eq` med et ekstra argument: `eq pr xs ys` returnerer `True` hvis for hver `x` i `xs` finnes en `y` i `ys` slik at `pr x y == True`, og for hver `y` i `ys` finnes en `x` i `xs` slik at `pr y x == True`. Det skal gjelde, f.eks., at `eq = eqG (==)` og
`eqG (<) [1,2,3] [3] = False`, mens `eqG (<=) [1,2,3] [3] = True`.

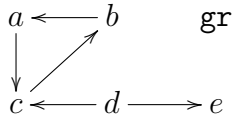
1.6. `ps::[t]->[[t]]` gir alle delmengder av argumentlisten (betraktet som en mengde). F.eks. (elementer i resultatlister kan komme i andre rekkefølger),

```
ps [] = [[]],
ps [1,2] = [[],[1],[2],[1,2]] = ps [1,2,1]
ps "abc" = ["","a","b","c","ab","ac","bc","abc"] = ps "baabbcc"
```

Det skal alltid gjelde at `mengde(ps xs) = True`.

2 Rettede grafer (35%)

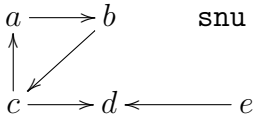
En (rettet) graf er en mengde av noder med kanter som forbinder utvalgte par av noder (i én retning, fra kilde- til målnode). F.eks.: følgende graf `gr` har 5 noder og 6 kanter:



noder: $[a, b, c, d, e]$
kanter, kL = $[(a, c), (b, a), (c, b), (d, c), (d, e)]$

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 en graf representeres som en *naboliste*, nemlig, en liste av par der første elementet er en node, si x, og andre er en liste av noder med kanter fra denne noden x. F.eks., representerer nabolisten `nL = [(a,[c]),(b,[a]),(c,[b]),(d,[c,e]),(e,[])]` den samme grafen `gr`. (Elementer uten utgående kanter, som `(e,[])`, kan men trenger ikke være med.)

2.1. ***Bruk mønstre*** for å programmere funksjon `snuK::[(t,t)]->[(t,t)]` som snur alle kanter i en graf gitt som en kantliste. Den skal snu hver kant uten å endre rekkefølgen på kanter. F.eks., `snuK kL` skal gi listen `[(c,a),(a,b),(b,c),(c,d),(e,d)]`, men ikke f.eks. `[(a,b),(c,a),(b,c),(c,d),(e,d)]`, selv om begge tilsvarer den samme snudde grafen:



2.2. Programmer funksjon `snuN::[(t,[t])]->[(t,[t])]` som snur alle kanter i en graf gitt som en naboliste. F.eks., `snuN nL = [(a,[b]),(b,[c]),(c,[a,d]),(e,[d])]`.

2.3. En sti er en liste med noder $[x_1, x_2, ..., x_n]$ der hvert par (x_i, x_{i+1}), for $1 \leq i < n$, er en kant. (En sti følger kanter kun i pilens retning, dvs., kun fra kilde- til målnoden.)

Bruk nabolisterepresentasjon for å programmere funksjon `reach::[(t,[t])]->t->[t]` (samt alle hjelpefunksjoner), slik at `reach gr x` returnerer en liste uten repetisjoner med alle noder (inklusiv `x`) som kan nåes fra inputnoden `x` ved å følge en eller annen sti i grafen `gr`. F.eks., for grafer `gr` og `snuN gr` (tilsvarende grafen `snu gr`) over,

```
reach gr c = [b,a,c]          reach (snuN gr) c = [b,a,c,d]
reach gr d = [d,e,c,b,a]      reach (snuN gr) d = [d]
reach gr e = [e]              reach (snuN gr) e = [e,d]
```

Rekkefølgen i resultatlister kan bli annerledes, men de skal være `eq` (oppgave **1.4.**) med lister fra eksemplene over, f.eks., `eq [b,a,c] (reach gr c) = True`, osv.. Husk å sikre terminering i alle tilfeller.

3 Litt IO (15%)

Programmer en funksjon (samt alle hjelpefunksjoner) `main::IO ()` som leser én linje input fra terminalen om gangen. Input kan være av tre typer:

1. Hvis input er tom, dvs. brukeren har trykket kun RETURN-knappen, avsluttes programmet.

Ellers består input av en sekvens av ord (ikke-tomme strenger adskilt med mellomrom):

2. Hvis input ikke starter med 1-bokstav ordet "L", behandles hele inputsekvensen av ord `x1 x2...xn` som beskrevet under.

3. Hvis input starter med ordet "L" brukes andre inputordet som filnavn; programmet leser da innholdet fra filen (som kan antas å eksistere) og behandler det som en sekvens `x1 x2...xn` av ord adskilt med mellomrom.

Hvis input er av den andre eller tredje typen, dvs. programmet har lest inn (fra terminalen, i punkt 2, eller fra en fil i punkt 3) en sekvens av ord `x1 x2...xn`, sjekker programmet først om alle ord er forskjellige. Hvis ikke, gir det en passende melding til brukeren og venter på neste kommando, mens hvis ja, så skriver det ut en liste med alle delmengder av inputlisten (som i oppgave **1.6.**), én i hver linje, og deretter venter på neste kommando.

4 Resonnering om programmer (25%)

4.1. Bevis at din implementasjon av `snuK` fra **2.1.** er slik at for enhver kantliste `ks` (dvs. liste av par `ks::Eq t => [(t,t)]`) gjelder likheten `snuK(snuK ks) = ks`.

4.2. Bevis at din implementasjon av `mengde` og `rep` i **1.1.** og **1.2.** tilfredstiller likheten `mengde(rep xs) = True` for enhver liste `xs`.

Bevisene bør bli komplette, dvs. hvis du bruker noen hjelpeligninger (f.eks., for hjelpefunksjoner) bør disse også bevises. Dersom du har dårlig tid, spesifiser i det minste hvilke hjelpeligninger du forutsetter i ditt hovedbevis.

[Hint: Dersom koden benytter if-uttrykk, kan følgende bli relevant. For en anvendelse av en funksjon `fun` på et if-uttrykk, gjelder det at:
`fun(if C then X else Y)` = `if C then (fun X) else (fun Y)`.
En likning `Z = if C then X else Y` kan generelt vises ved to tilfeller:
– under antakelse av `C=True` viser man `Z = X`, og
– under antakelse av `C=False` viser man `Z = Y`.]

Lykke til!
Michał Walicki

Problem 1 – solution

2.5 **1.1.** mengde [] = True
mengde (x:xs) = not (elem x xs) && mengde xs

2.5 **1.2.** rep [] = []
rep (x:xs) = if elem x xs then rep xs else x:rep xs

2.5 **1.3.** del [] _ = True
del (x:xs) ys = elem x ys && del xs ys

2.5 **1.4.** eq xs ys = del xs ys && del ys xs

5 **1.5.** eqG pr xs ys = delG pr xs ys && delG pr ys xs
delG pr [] _ = True
delG pr (x:xs) ys = or [pr x y | y <- ys] && delG pr xs ys

10 **1.6.** ps xs = pl (rep xs)
pl [] = [[]]
pl (x:xs) = [s | r <- pl xs, s <- [x:r,r]]
– – andre ligningen for pl kan bli annerledes, f.eks.:
pl (x:xs) = [if b then x:r else r | b <- [True, False], r <- pl xs]
= map (x:) (pl xs) ++ pl xs
= concat [[x:r,r] | r <- pl xs]

Problem 2 – solution

3 **2.1.** snuK kl = [(y,x) | (x,y) <- kl]
o.l. gir full uttelling – mønstre letter kun bevis i Oppgave 4;
snuK [] = [] og snuK ((x,y):rest) = (y,x) : snuK rest

16 **2.2.** Mange muligheter, f.eks.:
snuN xs = let kL = convNK xs in map (collectOne kL) (noder kL)
convNK nl = concat (map oneN nl) – – kantliste (med snudde kanter) utfra naboliste
oneN (a,ns) = map (\x -> (x,a)) ns – – liste med snudde kanter fra a -> x ∈ nl
noder kls = rep ([x | (x,z) <- kls] ++ [z | (x,z) <- kls])
collectOne kl x = (x,[z | (y,z) <- kl, y==x]) – – faa ut-naboer fra kantliste kl for x

16 **2.3.** naboer :: Eq t => [(t,[t])] -> t -> [t]
naboer nL x = let ls = [el | el <- nL, (fst el) == x] in
if null ls then [] else snd (head ls)
reach nL x = rep (concat (trav nL [] x))
trav nL vis x = if (elem x vis) then [vis]
else let rec = naboer nL x in
if null rec then [x:vis]
else concat (map (trav nL (x:vis)) rec)

Problem 3 – solution

10 for riktig IO oppsett (aksjoner, filinnlesing, do, osv.)
5 for programmering

```
main = do
   putStr "liste / CR: "
   cc <- getLine
   let c = words cc
   if null c then return ()
   else if (head c == "L") then do
     inh <- readFile (head (tail c))
     vis (words inh)
     main
   else do
     vis c
     main
vis x = if (mengde x) then skriv (ps x)
        else putStrLn "Listen er ikke en mengde."
skriv xs = mapM_ putStrLn (map show xs)
```

Problem 4 – solution

4.1. Basis følger ved første ligning for snuK: snuK (snuK []) $\stackrel{1}{=}$ snuK [] $\stackrel{1}{=}$ [] . 10

Antar IH: snuK(snuK r) = r, og viser snuK(snuK (x,y):r) = (x,y):r.

snuK (snuK ((x,y):r))	$\stackrel{2}{=}$	snuK ((y,x) : snuK r)
	$\stackrel{2}{=}$	(x,y) : snuK(snuK r)
	$\stackrel{IH}{=}$	(x,y) : r

4.2. Basis: mengde (rep []) $\stackrel{r1}{=}$ mengde [] $\stackrel{m1}{=}$ True 15

IH: mengde (rep xs) = True

mengde (rep (x:xs))	$\stackrel{r2}{=}$	mengde (if (elem x xs) then rep xs else x:rep xs)
Hint	=	if (elem x xs) then mengde(rep xs) else mengde(x:rep xs)
	$\stackrel{IH}{=}$	if (elem x xs) then True else mengde(x:rep xs)
hvis (elem x xs)=True	så	er det siste uttrykket =True; vi må dermed kun vise at
hvis (elem x xs)=False	så	mengde(x:rep xs) = True. Vi fortsetter da med:
		mengde(x:rep xs) =
	$\stackrel{m2}{=}$	not(elem x (rep xs)) && mengde (rep xs)
	$\stackrel{aux}{=}$	not(elem x xs) && mengde (rep xs)
not(elem x xs)=not False=True	=	True && mengde (rep xs)
(True && X) = X	=	mengde (rep xs)
	$\stackrel{IH}{=}$	True

Vi må bevise hjelpesetning aux: elem x xs = elem x (rep xs).

Basis for xs = [] : elem x (rep []) $\stackrel{r1}{=}$ elem x [], gir IH: elem x xs = elem x (rep xs).

$$\begin{aligned} \text{elem x (rep y:xs)} &\overset{r2}{=} \text{elem x (if elem y xs then rep xs else y:rep xs)} \\ &= \text{if elem y xs then elem x (rep xs) else elem x (y:rep xs),} \end{aligned}$$

som gir to tilfeller:

(1) elem y xs = True, da: $\begin{aligned} \text{elem x (rep y:xs)} &\overset{r2}{=} \text{elem x (rep xs)} \\ &\overset{IH}{=} \text{elem x xs} \\ &= \text{x==y || elem x xs} \\ &\overset{el}{=} \text{elem x (y:xs).} \end{aligned}$

Den nest siste likningen over følger ved å betrakte to mulige tilfeller:

if (x==y)=True then elem x xs = elem y xs = True, mens

if (x==y)=False then x==y || elem x xs = False || elem x xs = elem x xs.

(2) elem y xs = False og da: $\begin{aligned} \text{elem x (rep y:xs)} &\overset{r2}{=} \text{elem x (y:rep xs)} \\ &\overset{el}{=} \text{x==y || elem x (rep xs)} \\ &\overset{IH}{=} \text{x==y || elem x xs} \\ &\overset{el}{=} \text{elem x (y:xs).} \end{aligned}$

Kravene om *bruk av mønstre* var kun for å gjøre bevisene lettere, så man trekker ingenting for disse besvarelsene, hvis de er korrekte men ikke bruker mønstre. (Kandidaten "betaler" for å ikke ha brukt mønstre der, i oppgave 4.)