UNIVERSITETET I BERGEN

Det matematisk-naturvitenskapelige fakultet

NORSK

Eksamen i	:	INF-121 Programmeringsparadigmer
Dato	:	25 Februar 2016
Tid	:	9:00 – 12:00
Antall sider	:	3
Tillatte hjelpemidler	:	ingen

- Løsninger av delproblemer som du ikke har besvart kan antas gitt dersom de trenges i andre delproblemer.
- Forklar kort funksjoner/predikater som du selv innfører – i Haskell, angi deres type.
- Haskell-kode skal ikke bruke noen andre moduler enn *Prelude*.
- Prosentsatsene ved hver oppgave angir *kun omtrentlig* vekting ved sensur og forventet tidsforbruk/vanskelighetsgrad ved løsning.

1 Grammatikk (10%)

Vi har følgende grammatikk, med startsymbolet T, som tillatter oss å avlede uttrykk som representerer alle ikke-negative tall (dvs., 1, 2, 3, 145, osv.).

1. T ::= $D \mid TT$
2. D ::= $0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

1.1. Er uttrykket 001 med i språket definert av grammatikken? Hvis nei, forklar kort hvorfor og hvis ja, gi en venstre (leftmost) derivasjon av uttrykket.

1.2. Vis at grammatikken er tvetydig.

1.3. Skriv om grammatikken slik at den blir entydig og definerer det samme språket.
Tegn et parsetre for 145 i din grammatikk.

2 Haskell: typeavledning (20%)

Bruk algoritmen til Hindley-Milner, samt Martelli-Montanaris unifikasjonsalgoritme (begge skissert på slutten av settet), for å bestemme typen til følgene Haskell uttrykket eller vise at det ikke har noen type i Haskell: `\f -> \g -> \x -> f (g x)`.

Angi deretter typen til funksjonen `foo` definert ved følgende likningen, eller forklar hvorfor den ikke har noen type: `foo f g = \x -> f (g x)`.

3 Haskell: programmering (30%)

Vi skal håndtere en polymorf datatype bag i Haskell representert med lister, der rekkefølgen av elementene *ikke* spiller noen rolle, men antall forekomster gjør det. F.eks., `[1,2,1,2,2]` er en bag med 2 forekomster av 1 og 3 av 2; den samme bagen er, f.eks., listen `[2,2,2,1,1]`. Programmer i Haskell følgende funksjoner (og alle eventuelle hjelpefunksjoner):

3.1. `mult::Eq t => t -> [t] -> Int` – som returnerer antallet forekomster av det første argumentet i listen (bagen) oppgitt som det andre argumentet. F.eks., for l = bagen fra eksempelet over, ville vi fått `mult 1 l = 2`, `mult 2 l = 3` og `mult 5 l = 0`.

3.2. `maxel::Eq t => [t] -> (t,Int)` – som returnerer paret `(x,n)` der `x` er et element som forekommer flest, nemlig `n`, ganger i argumentlisten. F.eks., for bagen fra eksempelet, ville vi

fått `maxel l = (2,3)`.
NB! Elementet `x` trenger ikke å være entydig. F.eks., `maxel [1,2,1,2]` kan returnere `(1,2)` eller `(2,2)` – si hvordan din funksjon foretar et valg.

3.3. `eq::Eq t => [t] -> [t] -> Bool` – som returnerer `True` kun når begge argumentlister representerer den samme bagen, dvs., inneholder nøyaktig samme elementene, hvert med det samme antallet forekomster, f.eks.
`eq [1,2,1,2,2] [2,2,2,1,1] = True`, mens
`eq [1,2,1,2,2] [2,2,1,1] = False`.

3.4. `snitt::Eq t => [t] -> [t] -> [t]` som returnerer bagen tilsvarende snittet av de to argumentene, dvs. en bag der hvert element forekommer like mange ganger som i den argumentbagen som inneholder færrest forekomster av det, f.eks.,
`snitt [1,1,1,2,2,3,3] [1,1,2,3,3,3,4] = [1,1,2,3,3]`
`snitt [1,2,2,2,3,3,3] [2,4,2,4,4] = [2,2]`

3.5. `subsum::[Int] -> Int -> [[Int]]` som returnerer alle delbagene av inputsekvensen, hvis elementene summeres til tallet oppgitt som det andre argumentet, f.eks.,
`subsum [1,2,3,4,5] 7 = [ [3,4], [2,5], [1,2,4] ]`
`subsum [2,3,4] 8 = []`
`subsum [1,1,1,2] 3 = [ [1,1,1], [1,2], [1,2], [1,2] ]`
Det er nok at hvert mulig resultat skrives bare en gang, slik at i det siste tilfelle skal man helst få bare `[[1,1,1], [1,2]]`.

4 Prolog (40%)

"Definer predikat" betyr å programmere et Prolog predikat samt alle hjelpepredikater. Du kan bruke alle innebyggede predikater fra standard SWI-Prolog. Notasjon `pred(..+A..)` betyr at argumentet `A` antas instansiert ved kall av `pred`, mens mangel på en pluss, `pred(..B..)`, at argumentet må også kunne genereres ved et kall til `pred`.

Vi betrakter lister som representasjon av bagene, som beskrevet i oppgave 3. Du kan anta at argumenter som antas oppgitt ved kallet (merket med `+`) ikke inneholder noen variabler. Oppgaven er å definere følgende Prolog predikater, tilsvarende funksjoner fra oppgave 3.

4.1. `mult(+X,+L,N)` holder hvis N er antallet forekomster av elementet X i listen L.

4.2. `maxel(+L,X,N)` holder hvis og bare hvis X er et element som forekommer maximalt antall ganger, nemlig N, i listen L.

4.3. `eq(+L,+R)` holder hvis listene L og R representere den samme bagen.

4.4. `snitt(+L,+R,S)` holder hvis S er snittet av bagene L og R.

4.5. `subsum(+L,+N,S)` holder hvis S er delbagen av L (som inneholder tall), og summen av elementene i S er N.

Lykke til!
Michał Walicki

Hindley-Milner (a, b er ferske variabler):

(t1)	$E(\Gamma \mid con :: t)$	$= \{t = \theta(con)\}$	– for en konstant con
(t2)	$E(\Gamma \mid x :: t)$	$= \{t = \Gamma(x)\}$	– for en variabel x
(t3)	$E(\Gamma \mid f\ g :: t)$	$= E(\Gamma \mid g :: a) \cup E(\Gamma \mid f :: a \to t)$	
(t4)	$E(\Gamma \mid \backslash x \to ex :: t)$	$= \{t = a \to b\} \cup E(\Gamma, x :: a \mid ex :: b)$	

Martelli-Montanari:

$input$	$\Rightarrow resultat$	$applikasjonsbetinglese:$
$E, t = t$	$\Rightarrow E$	
$E, f(t_1 ... t_n) = f(s_1 ... s_n)$	$\Rightarrow E, t_1 = s_1, ..., t_n = s_n$	
$E, f(t_1 ... t_n) = g(s_1 ... s_m)$	$\Rightarrow NO$	$f \neq g$ $eller$ $n \neq m$
$E, f(t_1 ... t_n) = x$	$\Rightarrow E, x = f(t_1 ... t_n)$	
$E, x = t$	$\Rightarrow E[x/t], x = t$	$x \notin Var(t)$
$E, x = t$	$\Rightarrow NO$	$x \in Var(t)$

INF-121

Problem 1 – solution

1.1. Ja: T → TT → DT → 0T → 0TT → 0DT → 00T→ 00D → 001

1.2. Parsetreet for derivasjonen over bli annerledes enn for derivasjonen som starter med:
T → TT → TD → TTD → ...

1.3. Erstatt produksjonen T → TT med T → DT (eller T → TD).

Problem 2 – solution

kontekst	*uttrykk*	*til unifikasjon*
$\emptyset$	$\lambda f \to \lambda g \to \lambda x \to f(g\ x) :: t$	
$f :: r$	$\lambda g \to \lambda x \to f(g\ x) :: s$	$t = r \to s$
$f :: r, g :: q$	$\lambda x \to f(g\ x) :: p$	$s = q \to p, t = r \to s$
$f :: r, g :: q, x :: a$	$f(g\ x) :: b$	$p = a \to b, s = q \to p, t = r \to s$
$f :: r, g :: q, x :: a$	$f :: c \to b\ \&\ (g\ x) :: c$	$p = a \to b, s = q \to p, t = r \to s$
$f :: r, g :: q, x :: a$	$(g\ x) :: c$	$r = c \to b, p = a \to b, s = q \to p, t = r \to s$
$f :: r, g :: q, x :: a$	$x :: d\ \&\ g :: d \to c$	$r = c \to b, p = a \to b, s = q \to p, t = r \to s$
$f :: r, g :: q, x :: a$	$g :: d \to c$	$d = a, r = c \to b, p = a \to b, s = q \to p, t = r \to s$
$f :: r, g :: q, x :: a$		$q = d \to c, d = a, r = c \to b, p = a \to b, s = q \to p, t = r \to s$

unifikasjon (understrekket ligning substitueres i neste linjen) :

$$q = d \to c, \underline{d = a}, r = c \to b, p = a \to b, s = q \to p, t = r \to s$$

$$\underline{q = a \to c}, d = a, \underline{r = c \to b}, \underline{p = a \to b}, s = q \to p, t = r \to s$$

$$q = a \to c, d = a, r = c \to b, p = a \to b, \underline{s = (a \to c) \to (a \to b)}, t = (c \to b) \to s$$

$$q = a \to c, d = a, r = c \to b, p = a \to b, s = (a \to c) \to (a \to b), t = (c \to b) \to (a \to c) \to (a \to b)$$

Svaret er altså: $t = (c \to b) \to (a \to c) \to a \to b$ – og det er typen til foo.

Problem 3 – solution

Vi skriver hele programmet i et:

```
-- 2.1
mult x l = length [z | z<-l, z==x]
-- 2.2
maxel [x] = (x,1)
maxel (x:xs) = let els = elems (filter (/= x) xs) in maxi els (x:xs) (x,mult x (x:xs))

maxi [] ls (a,n) = (a,n)
maxi (x:xs) ls (a,n) = let mx = (mult x ls) in if mx > n then maxi xs ls (x,mx)
                                                 else maxi xs ls (a,n)
elems [] = []
elems (x:xs) = if (elem x xs) then elems xs else x: elems xs
-- 2.3
eq [] [] = True
eq [] (x:xs) = False
```

```
eq (x:xs) [] = False
eq (x:xs) ys = mult x (x:xs) == mult x ys && eq (filter (/= x) xs) (filter (/= x) ys)
-- 2.4
snitt [] _ = []
snitt (x:xs) ys = if elem x ys then x:snitt xs (reme x ys)
              else snitt xs ys
-- fjern bare en forekomst av elementet fra listen:
reme x [] = []
reme x (y:ys) = if x==y then ys else y:reme x ys
-- 2.5
subsum l s = remdup [x | x <- subseqs l, sum x == s]
subseqs [] = [[]]
subseqs (x:xs) = let ys = subseqs xs in ys ++ (map (x:) ys)
remdup [] = []
remdup (x:xs) = x : remdup (filter (not eq x) xs)
```

Problem 4 – solution

Vi skriver hele programmet i et og bruker et hjelpepredikat:`elems(S,E):- setof(T,member(T,S),E).`

```
%% 4.1
mult(X,[],0).
mult(X,[X|T],N1) :- mult(X,T,N), N1 is N+1.
mult(X,[Y|T],N) :- not(X=Y), mult(X,T,N).
%% 4.2
maxel([],_,0).
maxel([H|L],H,N) :- mult(H,[H|L],N), maxel(L,_,Z), N >= Z.
maxel([H|L],R,Z) :- mult(H,[H|L],N), maxel(L,R,Z), N < Z.
%% 4.3
eq([],[]).
eq([H|T],R) :- reme(H,R,Rr), eq(T,Rr).
% fjern bare en forekomst av elementet fra listen:
reme(H,[H|T],T) :- !.
reme(H,[X|T],[X|R]) :- H\=X,reme(H,T,R).
%% 4.4
snitt([],B,[]).
snitt([X|T],B,[X|Rest]) :- member(X,B), !, reme(X,B,Br), snitt(T,Br,Rest).
snitt([X|T],B,Rest) :-  snitt(T,B,Rest).
%% 4.5
subsum([],0,[]).
subsum([H|T],N,[H|R]):- NewN is N-H, NewN >= 0, subsum(T,NewN,R).
subsum([_|T],N,R) :- subsum(T,N,R).
```