Universitetet i Bergen
Det matematisk-naturvitenskapelige fakultet

Eksamen i	:	INF-121A Funksjonell programmering
Dato	:	21 February 2012
Tid	:	9:00 – 12:00
Antall sider	:	2
Tillatte hjelpemidler	:	ingen

- Løsninger av delproblemer som du ikke har besvart kan antas gitt dersom de trenges i andre delproblemer.
- Korrekte løsninger er desto bedre, jo kortere de er. Forklar kort funksjoner som du selv innfører.
- Prosentsatsene ved hver oppgave angir *kun omtrentlig* vekting ved sensur og forventet tidsforbruk/vanskelighetsgrad ved løsning.

1 Evaluering av postfiks uttrykk (40%)

Vi betrakter følgende gramatikk for aritmetiske uttrykk i postfiks notasjon:

```
E ::= Pos | E E * | E E + | E E -
Pos ::= Digit | Digit Pos          Digit ::= 0 | 1 | ...  8 | 9
```

der +, * og – betegner vanlige, binære aritmetiske operatorene og `Pos` er ikke-terminal symbol for ikke negative heltall (altså, uten noen minus tegn foran). Betydningen bestemmes ved at en binær operator anvendes på to uttrykk *til venstre* for den, med uttrykket lenger til venstre som det første argumentet, f.eks.:

(a) "11 2 3 * + 4 –" tilsvarer ((2*3)+11)–4
(b) "4 11 2 3 * + –" tilsvarer 4–((2*3)+11)

Postfiks uttrykk evalueres lett online (mens de leses fra venstre til høyre) ved å bruke en stabel. I det man leser et tall, legges det øverst på stabelen, mens i det man leser en operator, hentes to øverste tall fra stabelen, anvendes operator på dem, og så legges resultatet øverst på stabelen. Eksemplene viser stabelen i det token som står under den ble lest.

			3				
		2	2	6		4	
	11	11	11	11	17	17	13
(a)	11	2	3	*	+	4	–

				3			
			2	2	6		
		11	11	11	11	17	
	4	4	4	4	4	4	–13
(b)	4	11	2	3	*	+	–

Programmer en Haskell funksjon `posteval::String->[String]->Int` (helst uten noen hjelpefunksjoner) som evaluerer slike postfiks uttrykk, gitt som streng i første argumentet, ved å bruke stabel `[String]` som forklart over. Funksjonen kalles initielt med tom stabel og returnerer tallet som ligger på toppen av stabelen ved avsluttet evaluering, f.eks.,

(a) `posteval "11 2 3 * + 4 -" []` skal gi `13`, mens
(b) `posteval "4 11 2 3 * + -" []` skal gi `-13`.

2 Funksjoner (60%)

La `k,m` betegne to vilkårlige Haskell Eq-typer (muligens `k` = `m`). En "par-funk" `f` (partiell funksjon f, fra `k` til `m`) er en liste av par, `f::[(k,m)]`, som oppfyller følgende funksjonskravet: for vilkårlige to par $(k1, m1), (k2, m2) \in f$, hvis $k1 = k2$ så også $m1 = m2$. ($m1$ er da verdien av par-funken f i punktet $k1$, skrevet $f(k1) = m1$.) Mengden av k'er som forekommer i første posisjoner er da par-funkens domene, $dom(f)$, og mengden av m'er i andre posisjoner dens bilde, $bde(f)$ – i Haskell: `dom(f) = [k|(k,m)<-f]` og `bde(f) = [m|(k,m)<-f]`.

2.1. Lister kan ha multiple forekomster av samme elementer, men de sammenlignes i denne oppgaven som mengder, dvs. posisjon og repetisjon av elementer ikke spiller noen rolle. Definer en Haskell funksjon

```
seteq::Eq t => [t]->[t]->Bool
```

som for en vilkårlig Eq-type `t`, gir `True` dersom to input lister inneholder de samme elementer, og `False` ellers. F.eks. `seteq [1,2,4,2] [1,4,2] == True`, mens `seteq [1,2,4,2] [1,2] == False`.

2.2. Definer

Haskell funksjon	– som returnerer
`parFunk::[(k,m)]->Bool`	– `True` hvis og bare hvis input listen er en par-funk
`ev::[(k,m)]->k->m`	– `ev f x` gir en `error` melding dersom `f` ikke er en par-funk eller hvis $x \notin dom(f)$, og ellers gir verdien $f(x)$.

F.eks. (anta at `a,b` er definerte verdier)

`parFunk [(1,a),(2,b),(4,a),(2,b)] == True` og

`ev [(1,a),(2,b),(4,a),(2,b)] 2 == b`, mens

`parFunk [(1,a),(2,b),(4,a),(2,a)] == False`, siden to par med `2` i første posisjon og forskjellige verdier i andre posisjon strider mot funksjonskravet.

2.3. Definer en Haskell funksjon

```
comp::[(k,m)]->[(m,n)]->[(k,n)]
```

som tilsvarer sammensetting, dvs. for par-funker `f,g` med `seteq bde(f) dom(g) == True`, gir `comp f g` en par-funk `h::[(k,n)]` tilsvarende f etterfulgt av g (sammensetting av f og g), dvs. med `dom(h)=dom(f)` og slik at for hvert element $x \in dom(f) : h(x) = g(f(x))$. (Dersom `seteq bde(f) dom(g) == False`, skal `comp f g` gi en `error` melding.)

2.4. Definer en Haskell funksjon

```
rev::[(k,m)]->[(m,k)]
```

som, for et argument `f`, returnerer en liste med reverserte alle par fra `f`, dvs. `rev f` inneholder et par (b, a) hvis og bare hvis (a, b) er med i listen `f`. (Merk at `rev f` ikke trenger å vare en par-funk, selv om `f` er det.) Definer så en Haskell funksjon

```
inj::[(k,m)]->Bool
```

som gir `True` hvis og bare hvis input er en injektiv par-funk, dvs. slik at for alle $a, b \in dom(f)$, hvis $a \neq b$ så også $f(a) \neq f(b)$.

2.5. Identiteten på en liste `li`, er en par-funk `[(x,x)| x <- li]`. To par-funker `f::[(k,m)]` og `g::[(m,k)]` er *inbyrdes inverse*, dersom både

(i) f etterfulgt av g er identiteten på $dom(f)$, og

(ii) g etterfulgt av f er identiteten på $dom(g)$.

Definer en Haskell funksjon

```
inv::[(k,m)]->[(m,k)]->Bool
```

som returnerer `True` hvis og bare hvis argumenter er to par-funker som er innbyrdes inverse.

Lykke til!
Michał Walicki

Universitetet i Bergen
Det matematisk-naturvitenskapelige fakultet

Eksamen i	:	INF-121 Programmeringsparadigmer
Dato	:	21 February 2013
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.
- Korrekte løsninger er desto bedre, jo kortere de er. Forklar kort funksjoner som du selv innfører.
- Prosentsatsene ved hver oppgave angir *kun omtrentlig* vekting ved sensur og forventet tidsforbruk/vanskelighetsgrad ved løsning.

1 Haskell (45%)

La `k,m` betegne to vilkårlige Haskell Eq-typer (muligens `k = m`). En "par-funk" `f` (partiell funksjon f, fra `k` til `m`) er en liste av par, `f::[(k,m)]`, som oppfyller følgende funksjonskravet: for vilkårlige to par $(k1, m1), (k2, m2) \in f$, hvis $k1 = k2$ så også $m1 = m2$. ($m1$ er da verdien av par-funken f i punktet $k1$, skrevet $f(k1) = m1$.) Mengden av k'er som forekommer i første posisjoner er da par-funkens domene, $dom(f)$, og mengden av m'er i andre posisjoner dens bilde, $bde(f)$ – i Haskell: `dom(f) = [k|(k,m)<-f]` og `bde(f) = [m|(k,m)<-f]`.

1.1. Lister kan ha multiple forekomster av samme elementer, men de sammenlignes i denne oppgaven som mengder, dvs. posisjon og repetisjon av elementer ikke spiller noen rolle. Definer en Haskell funksjon

```
seteq::Eq t => [t]->[t]->Bool
```

som for en vilkårlig Eq-type `t`, gir `True` dersom to input lister inneholder de samme elementer, og `False` ellers. F.eks. `seteq [1,2,4,2] [1,4,2] == True`, mens `seteq [1,2,4,2] [1,2] == False`.

1.2. Definer

Haskell funksjon	– som returnerer
`parFunk::[(k,m)]->Bool`	– `True` hvis og bare hvis input listen er en par-funk
`ev::[(k,m)]->k->m`	– `ev f x` gir en `error` melding dersom `f` ikke er en par-funk eller hvis $x \notin dom(f)$, og ellers gir verdien $f(x)$.

F.eks. (anta at `a,b` er definerte verdier)

`parFunk [(1,a),(2,b),(4,a),(2,b)] == True` og
`ev [(1,a),(2,b),(4,a),(2,b)] 2 == b`, mens
`parFunk [(1,a),(2,b),(4,a),(2,a)] == False`, siden to par med `2` i første posisjon og forskjellige verdier i andre posisjon strider mot funksjonskravet.

1.3. Definer en Haskell funksjon

```
comp::[(k,m)]->[(m,n)]->[(k,n)]
```

som tilsvarer sammensetting, dvs. for par-funker `f,g` med `seteq bde(f) dom(g) == True`, gir `comp f g` en par-funk `h::[(k,n)]` tilsvarende f etterfulgt av g (sammensetting av f og g), dvs. med `dom(h)=dom(f)` og slik at for hvert element $x \in dom(f) : h(x) = g(f(x))$. (Dersom `seteq bde(f) dom(g) == False`, skal `comp f g` gi en `error` melding.)

1.4. Definer en Haskell funksjon

`rev::[(k,m)]->[(m,k)]`
som, for et argument `f`, returnerer en liste med reverserte alle par fra `f`, dvs. `rev f` inneholder et par (b, a) hvis og bare hvis (a, b) er med i listen `f`. (Merk at `rev f` ikke trenger å vare en par-funk, selv om `f` er det.) Definer så en Haskell funksjon
`inj::[(k,m)]->Bool`
som gir `True` hvis og bare hvis input er en injektiv par-funk, dvs. slik at for alle $a, b \in dom(f)$, hvis $a \neq b$ så også $f(a) \neq f(b)$.

1.5. Identiteten på en liste `li`, er en par-funk `[(x,x)| x <- li]`. To par-funker `f::[(k,m)]` og `g::[(m,k)]` er *inbyrdes inverse*, dersom både
(i) f etterfulgt av g er identiteten på $dom(f)$, og
(ii) g etterfulgt av f er identiteten på $dom(g)$.
Definer en Haskell funksjon
`inv::[(k,m)]->[(m,k)]->Bool`
som returnerer `True` hvis og bare hvis argumenter er to par-funker som er innbyrdes inverse.

2 Prolog (45%)

"Definer predikat" betyr å programmere Prolog predikatet samt alle hjelpepredikater. Lister av par antas representert som i Haskell, dvs. med syntaks $[(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)]$. 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`.

Oppgaven er å programmere i Prolog predikater tislvarende funksjoner fra oppgave 1.

2.1. Definer predikat `seteq(+A,+B)` som holder hvis to input lister inneholder de samme elementer. F.eks., `seteq([1,2,1,1],[2,1])` holder og `seteq([1,2,3],[1,2,2])` holder ikke.

2.2. Definer predikat `parFunk(+F)` som holder hvis input listen er en par-funk. Definer så predikater:

1. `dom(+F,D)`: holder dersom `D` er domenet til par-funken `F`,
2. `bde(+F,B)`: holder dersom `B` er bildet til par-funken `F`,
3. `ev(+F,+A,V)`: holder hvis `F` er en par-funk med verdien `V` i punktet `A` (dvs. $F(A) = V$).

2.3. Definer predikat `comp(+F,+G,H)` som holder hvis `H` er en par-funk tilsvarende sammensetting av par-funker `F` og `G`, dvs. når `bde(F,B)`, `dom(G,D)` og `seteq(B,D)` holder, så for hver $x \in dom(F)$: `ev(H,x,r)` holder hvis og bare hvis det finnes en $b \in bde(F)$ slik at `ev(F,x,b)` og `ev(G,b,r)` holder (i matematisk notasjon, for alle $x \in dom(F) = dom(H)$: $H(x) = G(F(x))$.)

2.4. Definer predikat `rev(+A,B)` som holder dersom `A` er en liste av par, og `B` er listen med disse par reversert, dvs. der `(y,x)` forekommer i `B` hvis og bare hvis `(x,y)` forekommer i `A`.

Definer så et predikat `inj(+F)` som holder hvis `F` er en injektiv par-funk, dvs. slik at for alle $a, b \in dom(F)$, hvis $a \neq b$ så også $F(a) \neq F(b)$.

2.5. Definer predikat `erId(+F)` som holder hvis `F` er identiten dvs. en par-funk bestående utelukkende av par på formen `(X,X)` (for alle $x \in dom(F) : F(x) = x$.)

To par-funker `F` og `G` er *inbyrdes inverse*, dersom både
(i) `F` etterfulgt av `G` er identiteten på $dom(F)$, og

(ii) `G` etterfulgt av `F` er identiteten på $dom(G)$.

Definer et predikat

`inv(+F,+G)`

som holder hvis og bare hvis argumenter er to innbyrdes inverse par-funker.

3 Unifikasjon (10%)

Variablene er storebokstaver X,Y,Z. All unifikasjon under utføres med `occurs check`.

3.1. f(g(X),X) = f(Y,a) – Gi alle stegene og resultatet av unifikasjonsalgoritme (med `occurs check`) utført på denne ligningen.

3.2. f(g(X),X) = f(Y,Y) – Gi alle stegene og resultatet av unifikasjonsalgoritme (med `occurs check`) utført på denne ligningen.

3.3. f(g(X),X) = f(Y,Z) – Er substitusjon {X=a,Y=g(a), Z=a} et mulig resultat av unifikasjonsalgoritme kjørt på denne ligningen? Forklar kort svaret.

Lykke til!

Michał Walicki

INF-121A

Problem 1 – solution

Postfiks evaluering

```
import IO
import Data.Char

-- read the string (from left to right) pushing all
-- operands (non-negative Ints) on the stack while
-- encountering en operator, evaluate it with the two
-- top operands popped from the stack and push the result on the stack

postev "" (x:st) = read x ::Int
postev (' ':str) st = postev str st
postev (v:str) st = if isDigit v then
                     let (t,d) = span isDigit (v:str) in postev d (t:st)
                    else
                      postev str ((show r):cc) where
                      a=head st
                      bb=tail st
                      b=head bb
                      cc=tail bb
                      r = case v of
                        '*' -> (read a ::Int)*(read b ::Int)
                        '+' -> (read a ::Int)+(read b ::Int)
                        '-' -> (read b ::Int)-(read a ::Int)
                                --lower in stack is first
```

Problem 2 – solution

Funksjoner

```
dom f = [x|(x,y) <- f]
bde f = [y|(x,y) <- f]
```

2.1.

```
subset xs ys = all (\x -> elem x ys) xs --[elem x ys | x <- xs]
seteq x y = subset x y && subset y x
```

2.2.

```
parFunk [] = True
parFunk ((k,m):kms) = all (==m) [y|(x,y)<-kms,x==k] && parFunk kms

ev f x = if not(parFunk f) || not(elem x (dom f)) then error "Not funk or dom"
         else head [y|(a,y)<-f, a==x]
```

2.3.

```
rev f = [(y,x) | (x,y) <- f]
inj f = parFunk (rev f)
```

2.4.

```
comp f g = if seteq (bde f) (dom g) then
              [(a,ev g (ev f a))| a <- dom f]
           else error "Not composable"
```

2.5.

```
isid [] = True
isid ((x,y):xy) = if x==y then isid xy else False

inv f g = isid (comp f g) && isid (comp g f)
```

INF-121

Solution to exercise 1 is given in the solutions to INF-121A.

Problem 2 – solution

2.1. subset/2 er innebygget i Prolog

```
seteq(A,B) :- subset(A,B), subset(B,A).
```

2.2.

```
parfunk([]).
parfunk([(A,X)]).
parfunk([(A,X),(B,Z)|T]) :- not(A=B), !, parfunk([(A,X)|T]), parfunk([(B,Z)|T]).
parfunk([(A,X),(A,X)|T]) :- parfunk([(A,X)|T]).

ev(F,A,X) :- parfunk(F), ev1(F,A,X).
ev1([(A,X)|T],A,X):- !.
ev1([(A,X)|T],B,Y):- ev1(T,B,Y).
```

2.3.

```
comp(F,G,H) :- parfunk(F), parfunk(G), comp1(F,G,H).
comp1([],B,[]).
comp1([(A,X)|T],B,[(A,Y)|S]) :- ev1(B,X,Y), comp1(T,B,S).
```

2.4.

```
rev([],[]).
rev([(A,X)|T], [(X,A)|S]) :- rev(T,S).

inj(F) :- rev(A,B), parfunk(B).
```

2.5.

```
erid([]).
erid([(X,X)|S]) :- erid(S).

inv(A,B) :- comp(A,B,AB), comp(B,A,BA), erid(AB), erid(BA).
```

Problem 3 – solution

3.1. f(g(X),X) = f(Y,a) $\rightarrow$Y = g(X), X=a $\rightarrow$Y = g(a), X=a – dette er resulterende mgu.

3.2. f(g(X),X) = f(Y,Y)
Y = g(X), X = Y $\rightarrow$Y = g(Y), X=Y $\rightarrow$feiler grunnet occurs check

3.3. f(g(X),X) = f(Y,Z) – substitutsjonen er en unifikator men ikke mest generell sådan, og derfor vil den ikke produseres av Martelli-Montanari algoritme, som gir kun mgu.