Universitetet i Bergen

Det matematisk-naturvitenskapelige fakultet

Nynorsk

Eksamen i	:	INF-121 Programmeringsparadigmer
Dato	:	4 Desember 2015
Tid	:	9:00 – 12:00
Antall sider	:	3
Tillatte hjelpemidlar	:	ingen

- Løysningar av delproblem som du ikkje har besvart kan antas gitt dersom de trenges i andre delproblem.
- Forklar kort funksjonar/predikat som du sjølv innfører – i Haskell, angi deira type.
- Haskell-kode skal ikkje bruke nokon andre modular enn *Prelude*.
- Prosentsatsane ved kvar oppgåve angir *kun omtrentleg* vekting ved sensur og forventa tidsforbruk/vanskegrad ved løysning.

1 Haskell **(35%)**

Vi behandler følgjande prefiks grammatikk for boolske uttrykk:

```
BA := Tegn | - BA | * BA BA | + BA BA
Tegn := a | b | ...  | z | 0 | 1 (enkel bokstav eller 1, 0)
```

`1` representerer 'sann', `0` 'usann', `-` boolsk negasjon, `*` konjunksjon og `+` disjunksjon, mens småbokstaver variabler. F.eks., skal uttrykket `+ a - a` evaluere til `1` for alle tilordningar av `1/0` til variabelen `a`. Vi skal bruke følgjande datatypen for å representere abstrakte syntakstre for `BA`-uttrykkene:

```
data Ast = S | U | Var Char | Ik Ast | Og Ast Ast | El Ast Ast
                                                deriving (Eq,Show,Read),
```

der `S` tilsvarer 1 (sann) og `U` 0 (usann). `Ast` for uttrykket `+ a - 0` blir da `El Var 'a' Ik U`.

1.1. Definer Haskell funksjon `parse::String -> Ast` som parser input streng og returnerer resulterande Ast for inpututtrykket. Du kan anta at input er korrekt, men kall helst `error` funksjonen på passende stadar.

1.2. Definer ein funksjon `vars::Ast -> [Char]` som returnerer ei liste (uten duplikat) med alle variabelnamn som finst i uttrykket representert av input Ast. F.eks.

```
vars (parse ``+ a * a b'') = ``ab''.
```

1.3. Gitt ei liste med variabelnamn, definer ein funksjon `tilord::[Char] -> [[(Char,Ast)]]`, der det andre elementet i kvart par er anten `S` eller `U`, mens den resulterande lista representerer alle moglege tilordninger av boolske verdiar `S/U` til variablane i inputlista. (Du kan anta at inputlista ikkje inneheld multiple førekomstar av nokon teikn.) F.eks.

```
tilord ``ab'' = [ [('a',S),('b',S)], [('a',S),('b',U)],
                                 [('a',U),('b',S)], [('a',U),('b',U)] ]
```

1.4. Definer ein funksjon `eval::Ast -> [(Char,Ast)] -> Ast` som evaluerer uttrykket i det fyrste argumentet under variabeltilordning gitt i lista i det andre argumentet. F.eks.

```
eval (El Var 'a' Ik Var 'b') [('a',U),('b',S)] = U, mens
eval (El Var 'a' Ik Var 'b') [('a',U),('b',U)] = S.
```

`Ast` resulterande fra `eval` skal vere anten `S` eller `U`. (Negasjon er `S`(ann) viss og berre viss dens argument er U(sann), konjunksjon er S viss og berre viss begge argumenta er S, og disjunksjon er U viss og berre viss begge argumenta er U.)

1.5. Definer ein funksjon `sat::String->[[(Char,Ast)]]` som for ein inputstring tilsvarande eit BA-uttrykk, returnerer lista med alle tilordninger (av S eller U) til dets variablar, for hvilke uttrykket evaluerer til S(ann). F.eks.,

```
sat ``+ a - b'' = [[('a',S),('b',S)],[('a',S),('b',U)],[('a',U),('b',U)]],
sat ``* a - a'' = [].
```

Deretter definer ein funksjon `taut::String->Bool` som for ein inputstring tilsvarande eit BA-uttrykk, returnerer `True` viss det evaluerer til S under alle tilordningene til dets variablar, og `False` ellers, f.eks.

```
taut ``* a - a'' = False,
taut ``+ a - a'' = True.
```

2 Haskell: typeavledning (20%)

Bruk algoritmen til Hindley-Milner, samt Martelli-Montanaris unifikasjonsalgoritme (begge skissert på slutten av settet), for å bestemme typen til følgjande Haskell uttrykket eller vise at det ikkje har nokon type i Haskell:

```
\x -> \y -> x (y x).
```

3 Prolog (35%)

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

Vi behandler følgjande grammatikk for boolske uttrykk (litt endret i.fh.t. Oppgave **1**):

```
BA := Tegn | -(BA) | *(BA,BA) | +(BA,BA)
Tegn := a | b | ...  | z | 0 | 1 (enkel bokstav eller 1, 0)
```

med samme konvensjon som i Oppgave **1** (1 for 'sann', 0 'usann', - for negasjon, * konjunksjon og + disjunksjon.) F.eks., skal uttrykket `+(a,-(a))` evaluere til 1 for begge boolske verdiar tilordna til variabelen a.

3.1. Definer predikater `ikke(A,R)`, `og(A,B,R)` og `eller(A,B,R)`, som koder boolske tabellar for dei respektive boolske operatorane (som beskrevet i Oppgave **1.4.**). F.eks., `og(A,B,R)` holder viss og berre viss anten alle tre argumenta er 1, eller viss R og minst eit av A, B er 0; tilsvarande for `ikke` og `eller`. Kvar av A,B,R her er 0 eller 1.

3.2. Definer predikat `eval(+BA,+Env,V)`, som returnerer i V verdien av uttrykket BA evaluert under tilordning til dens variablar gitt ved Env. F.eks., `eval(+(-(a),b),[a:1,b:0],V)` skal gi `V=0.`, mens `eval(+(-(a),b),[a:1,b:1],V)` skal gi `V=1.`

[Hint: Predikatet

```
bokstav(V):- atom(V),atom_length(V,1),char_code(V,Kode),97=<Kode,Kode=<122.
```

kan nyttast for å sjekke om V er instansiert til ein enkel småbokstav.]

3.3. Definer predikat `sat(+BA,Env)` som held dersom uttrykket BA evaluerer til 1 under boolsk tilordning til dets variablar gitt ved Env (og som kan generere ein slik tilordning, dersom noen finst), og `taut(+BA)` som held viss uttrykket BA evaluerer til 1 under alle boolske tilordninger til dets variablar.

4 Søketre **(10%)**

Vi behandler følgjande Prolog program:

```
far(jan,per).
mor(jan,mari).
forelder(X,Y) :- far(X,Y).
forelder(X,Y) :- mor(X,Y).
```

4.1. Kva blir Prologs svar til spørringa (når ein trykker ; gjentatte gongar):

```
forelder(jan,X).
```

Tegn søketre for denne spørringa.

4.2. Vi erstatter den tredje klausulen i programmet over med følgende klausulen:

```
forelder(X,Y) :- far(X,Y),!.
```

Hva blir nå Prologs svar til spørringa (når man trykker ; gjentatte ganger):

```
forelder(jan,X).
```

Tegn søketre for denne spørringa.

Lykke til!
Michał Walicki

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

(t1)	$E(\Gamma \mid con :: t)$	$= \{t = \theta(con)\}$	– for ein konstant con
(t2)	$E(\Gamma \mid x :: t)$	$= \{t = \Gamma(x)\}$	– for ein 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

Vi skriver hele programmet i et:

```
-- import Data.List.nub -- for fjerning av duplikater fra en liste,
--                         men vi implementerer den selv

data Ast = S | U | Var Char | Ik Ast | El Ast Ast | Og Ast Ast
                                          deriving (Eq, Show, Read)

parse xs = fst (prs xs)
prs :: String -> (Ast,String)
prs ('0':xs) = (U,xs)
prs ('1':xs) = (S,xs)
prs ('+':xs) = let {(a,as) = prs xs; (b,bs) = prs as} in (El a b, bs)
prs ('*':xs) = let {(a,as) = prs xs; (b,bs) = prs as} in (Og a b, bs)
prs ('-':xs) = let (a,as) = prs xs in (Ik a, as)
prs (' ':xs) = prs xs
prs (x:xs) = (Var x,xs)

nub [] = []
nub (x:xs) = if (elem x xs) then nub xs else x:nub xs

vars xs = nub (varsRec xs)
varsRec S = []
varsRec U = []
varsRec (Var x) = [x]
varsRec (Ik e) = varsRec e
varsRec (El a b) = varsRec a ++ varsRec b
varsRec (Og a b) = varsRec a ++ varsRec b

tilord [] = [[]]
tilord (x:xs) = let al = tilord xs in [(x,S):s|s <- al] ++ [(x,U):s|s <- al]

get x ((a,v):xs) = if x==a then v else (get x xs)
eval S env = S
eval U env = U
eval (Var x) env = get x env
eval (Ik e) env = if (eval e env)==U then S
                  else U
eval (El a b) env = if ((eval a env)==S || (eval b env)==S) then S
                    else U
eval (Og a b) env = if ((eval a env)==S && (eval b env)==S) then S
                    else U
```

```
sat xs = let ba = parse xs in
             [as | as <- tilord (vars ba), (eval ba as)==S]
taut xs = let ba = parse xs in
              and [(eval ba as)==S | as <- tilord (vars ba)]
```

Problem 2 – solution

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

unifikasjon (understrekket ligning substitueres i samme linjen) :

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

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

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

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

Svaret er altså: $t = (c \to b) \to ((c \to b) \to c) \to b$.

Problem 3 – solution

Vi skriver hele programmet i et:

```
v(V) :- atom(V), atom_length(V,1),char_code(V,Kode), 97 =< Kode, Kode =< 122.
eval(V,[V:B|_],B) :- v(V).
eval(V,[W:_|Env],R) :- v(V), not(V=W), eval(V, Env, R).
eval(ik(AST),Env, R) :- eval(AST, Env, R1), ikke(R1,R).
eval(el(AST1,AST2),Env,R) :- eval(AST1,Env,R1), eval(AST2,Env,R2), eller(R1,R2,R).
eval(og(AST1,AST2),Env,R) :- eval(AST1,Env,R1), eval(AST2,Env,R2), og(R1,R2,R).
ikke(0,1).    ikke(1,0).
og(1,1,1).    og(1,0,0).
og(0,1,0).    og(0,0,0).
eller(0,0,0). eller(0,1,1).
eller(1,0,1). eller(1,1,1).

sat(P,X) :- eval(P, X, 1).
taut(P) :- \+ eval(P, _, 0).
```

Problem 4 – solution

4.1. Svaret blir `X=per` ; `X=mari`.

4.2. Her blir svaret `X=per.` – siden snitt eliminerer videre søk.