PK, 4. juni 2002

Løsningsforslag til DM22 eksamen juni 2002

Opgave 1 (20 %)

1.a: Ved kørsel med trace fås:

```
| ?- max([3,2,1],X).
      1    1  Call: max([3,2,1],_21) ?
      2    2  Call: max([2,1],_90) ?
      3    3  Call: max([1],_114) ?
      3    3  Exit: max([1],1) ?
      4    3  Call: 1>2 ?
      4    3  Fail: 1>2 ?
      2    2  Exit: max([2,1],2) ?
      4    2  Call: 2>3 ?
      4    2  Fail: 2>3 ?
      1    1  Exit: max([3,2,1],3) ?

X = 3 ?
      1    1  Redo: max([3,2,1],3) ?
      2    2  Redo: max([2,1],2) ?
      3    3  Redo: max([1],1) ?
      4    4  Call: max([],_138) ?
      4    4  Fail: max([],_126) ?
      3    3  Fail: max([1],_102) ?
      2    2  Fail: max([2,1],_78) ?
      1    1  Fail: max([3,2,1],_21) ?

no
```

1.b: En løsning der anvender en akkumulator er:

```
max(L,M) :- maxi(L,X,M).

maxi([],Max,Max).
maxi([H|T],X,Max) :- var(X),!, X = H, maxi(T,H,Max).
maxi([H|T],Sofar,Max) :- (H > Sofar, !, Newmax=H; Newmax=Sofar),
                         maxi(T,Newmax,Max).
```

Opgave 2 (25 %)

2.a: Der er to løsninger, afhængig af hvordan man fortolker proriteringerne af $\exists$ og $\wedge$, nedenfor illustreret ved paranteser. Ved kørsel af bogens algoritmer (rettede) fås følgende to løsninger, den første svarer til at $\exists X$ dækker alle forekomster af X, den sidste svarer til gprologs fortolkning uden at der er sat ekstra paranteser.

```
| ?- clausify(exists(X,p(X)->all(Y,p(Y)->p(f(X,Y)))&(all(Z,q(X,Z)->p(Z))))).
implout: exists(_15,(~p(_15)#all(_18,~p(_18)#p(f(_15,_18))))&all(_35,~q(_15,_35)#p(_35)
negin: exists(_15,(~p(_15)#all(_18,~p(_18)#p(f(_15,_18))))&all(_35,~q(_15,_35)#p(_35)))
skolem: (~p(f4)#all(_18,~p(_18)#p(f(f4,_18))))&all(_35,~q(f4,_35)#p(_35))
univout: (~p(f4)# ~p(_18)#p(f(f4,_18)))& ~q(f4,_35)#p(_35)
conjnf: (~p(f4)# ~p(_18)#p(f(f4,_18)))& ~q(f4,_35)#p(_35)
clausify: [cl([p(f(f4,_18))],[p(f4),p(_18)]),cl([p(_35)],[q(f4,_35)])|_310]

p(f(f4,_18)) :- p(f4), p(_18).
p(_35) :- q(f4,_35).

finished!!
(10 ms) yes
| ?- clausify(exists(X,p(X)->all(Y,p(Y)->p(f(X,Y))))&(all(Z,q(X,Z)->p(Z)))).
implout: exists(_15,~p(_15)#all(_18,~p(_18)#p(f(_15,_18))))&all(_38,~q(_15,_38)#p(_38))
negin: exists(_15,~p(_15)#all(_18,~p(_18)#p(f(_15,_18))))&all(_38,~q(_15,_38)#p(_38))
skolem: (~p(f6)#all(_18,~p(_18)#p(f(f6,_18))))&all(_38,~q(_15,_38)#p(_38))
univout: (~p(f6)# ~p(_18)#p(f(f6,_18)))& ~q(_15,_38)#p(_38)
conjnf: (~p(f6)# ~p(_18)#p(f(f6,_18)))& ~q(_15,_38)#p(_38)
clausify: [cl([p(f(f6,_18))],[p(f6),p(_18)]),cl([p(_38)],[q(_15,_38)])|_264]

p(f(f6,_18)) :- p(f6), p(_18).
p(_38) :- q(_15,_38).

finished!!
```

2.b: En mulig løsning er:

```
unifiable([],_,[]).

unifiable([First|Rest],Term,List):-
    not(First = Term), unifiable(Rest,Term,List).

unifiable([First|Rest],Term,[First|List]) :- unifiable(Rest,Term,List).
```

Opgave 3 (30 %)

3.a: Funktionen har signaturen:

```
 mpa ::(a -> b -> (a, c)) -> a -> [b] -> (a, [c])}.
```

Givet en funktion af to variable og et argument, anvendes funktionen på dette argument og successivt på alle elementerne i en liste, idet resultatet afleveres i en ny liste, parret sammen med afbilsningen af det oprindelige argument, fx.:

```
Main> mpa (\x y -> (2*x,x+y)) 1 [1,2,3]
(8,[2,4,7])
```

3.b: Første algoritme er lineær, og den anden er logaritmisk i `m`. Det samme gælder for pladskravet; men ved at omskrive til halerekursion kan det reduceres til et konstant pladskrav. Dog da Haskell er lazy skal der også indføres en akkumulator, eller sikres strictness på en eller anden måde.

3.c: Med følgende definitioner:

```
filter           :: (a -> Bool) -> [a] -> [a]
filter p []    = []
filter p (x:xs)
   | p x       = x : filter p xs
   | otherwise =     filter p xs

p &&& q = \x -> (p x && q x)
```

fås ved induktion for basetilfældet trivielt at

```
filter p (filter q []) = [] = filter (p &&& q) []
```

Så fremt `(p x && q x)` er sand for `x:xs` fås at

```
  filter p (filter q (x:xs))
= filter p (x : filter q xs)
= x : (filter p (filter q xs))
= x : (filter (p &&& q) xs)
```

per induktionsantagelsen. Såfremt `p x && q x` er falsk for `x:xs` må enten `p x` eller `q x` eller begge være falske. Hvis `p x` er falsk men `q x` er sand fås

```
  filter p (filter q (x:xs))
= filter p (x : filter q xs)
= (filter p (filter q xs))
= (filter (p &&& q) xs)
```

etc.

Opgave 4 (25 %)

Mulige løsninger:

4.a:
```
polAdd :: (Num a) => [a] -> [a] -> [a]

polAdd (a:as) (b:bs) =  (a+b) : polAdd as bs
polAdd as [] =  as
polAdd [] bs =  bs
```

4.b:
```
polEval :: (Num a) => [a] -> a -> a

polEval (a:as) x = a + x * polEval as x
polEval [] _ = 0
```

4.c:
```
polMul :: (Num a) => [a] -> [a] -> [a]

polMul as (b:bs) =  polAdd [a*b|a <- as] (polMul (0:as) bs)
polMul as [] = [0]
```