Rättningsmall i DD1361 Programmeringsparadigm, 15 dec 2009

Kursansvarig: Lars Arvestad

- Denna rättningsmall innehåller vägledning i rättningen. När du är lite osäker över hur du ska sätta poäng bör du först och främst använda ditt eget omdöme! Det brukar vara bra.
- Kom ihåg att det är tentandens uppgift att övertyga dig om att lösningen är korrekt. Du ska inte behöva anstränga dig för att förstå lösningen.
- Om du är mycket osäker på hur du ska rätta kan du göra en markering om det på försättsbladet och så gör Lasse en bedömning.
- Glöm inte att föra över poängen till försättsbladet! Var snäll och summera poängen!

Allmänna frågor

1. När typangivelse saknas, helt eller delvis, i ett program, kan en kompilator (ibland) lista ut vilken typ en variabel måste ha genom att studera dess sammanhang. Detta är en central finess i Haskell, men används även i exv Lisp.

   Exempel:

   ```
   function f(x):
      y = round(x)
      return y
   ```

   Om funktionen round bara är definierad på flyttal och alltid returnerar ett heltal blir x och y tvingade till att vara flyttals- respektive heltals-variabler. (2p)

2. Det krävs extra resurser för att "administrera" evalueringen. Det kan kosta både tid och minne. Tid, därför att man istället för att evaluera direkt sparar uttrycket tills att det behövs. Minne, därför att det krävs en liten datastruktur som beskriver vad som ska evalueras. (2p)

3. (a) Med semantik avser man *betydelsen* hos en instruktion som inte alltid är självklar. Tag till exempel en av inkrement-operatorerna i C-familjen: syntaxen ("notationen") är `i++` och semantiken är att variabeln `i` inkrementeras *efter* att värdet av `i` har använts, dvs värdet av uttrycket `i++` är värdet av `i` innan inkrementering. (1p)

   (b) Det betyder att en variabel inte kan anta ett otillåtet värde. (1p)

4. Här är några problem:

   - Man måste hitta ett sätt att översätta typer. Strängar i C skiljer sig från strängar i (exempelvis) Haskell. Numeriska typer kan också skilja sig åt. Boolska värden kan representeras på olika sätt.
   - Semantiken i funktionsanrop kan skilja sig. Exempelvis måste man komma ihåg att anropen till C-funktioner inte är lat evaluerade om man arbetar i Haskell, och sidoeffekter är inte tillåtna.
   - Språken kan hantera *exceptions* olika. C har ingen standardmekanism för exceptions.
   - En anropad C-funktion får inte lämna allokerat minne liggandes nånstans, utan måste fungera ihop med skräpsamlaren.

   (2p)

Funktionell programmering i Haskell

5. En poäng per typ.

   (a) `filter on :: Num a -> [a] -> [a]`

   (b) `annotate :: Int -> (Int,Bool)`

6. (a) `nr :: (Float -> Float) -> (Float -> Float) -> Float -> Float -> Float` (1p)

   (b) Lösningsförslag: (2p)

```
nr f fderiv precision x0 =
    let fval = f x0             -- f(x)
        deriv = fderiv x0       -- f'(x)
        x1 = x0 - fval / deriv  -- Beräkna nästa x
    in if (abs(x1 - x0) < precision) -- Konvergens?
       then x1
       else nr f fderiv precision x1  -- Iterera
```

7. Lösningsförslag: (3p)

```
type Pkt = (Int, Int)
boundingbox1 :: [Pkt] -> (Pkt, Pkt)
boundingbox1 l =
    let xl = map fst l
        yl = map snd l
        xmin = minimum xl
        xmax = maximum xl
        ymin = minimum yl
        ymax = maximum yl
    in ((xmin, ymin), (xmax, ymax))
```

8. Lösningsförslag: (4p)

```
search :: Eq a => a -> [Elem a] -> (Maybe Int, [Elem a])
search x [] = (Nothing, [Elem x 1]) -- Basfall
search x ((Elem y c) : t) =
    if (x == y)
    then let answer  = Just c   -- Vi har hittat ett svar
             newelem = Elem x (c + 1) -- Inkrementera räknaren
             newlist = (newelem : t) -- Bygg ihop ny lista
         in (answer, newlist)   -- Returnera
    else let (answer, t2) = search x t
         in (answer, (Elem y c) : t2) -- Bygg ihop ny lista
```

   Poängavdrag:

   - `Just` och `Nothing` inte har använts.
   - Basfall saknas
   - Felaktigt basfall
   - Räknaren inte inkrementeras
   - Den uppdaterade listan inte konstrueras på rätt sätt

Logikprogrammering med Prolog

9. Lösningsförslag:

```
positivt(X) :- X > 0.
```

Inget poäng om man returnerar svaret i en extra parameter. (1p)

10. Exempellösning: (3p)

```
union([],L2,L2).
union([H|T], L2, Lut) :-
     member(H,L2), !, union(T, L2, Lut).
union([H|T], L2, [H|Lut]) :-
     union(T, L2, Lut).
```

Poängavdrag göres bl.a. då

- tomma listan inte hanteras på ett bra sätt,
- upprepningar av element tillåts,
- predikatet kan generera felaktiga lösningar vid upprepat användande.

11. (a) Exempellösning: (1p)

```
genereramera(L, X) :- member(X, L).
```

(b) Exempellösning:

```
genereramera(L, X) :- member(X, L), X<0, !, fail.
genereramera(L, X) :- member(X, L).
```

Först testar vi om det finns ett negativt element, och i så fall sätter vi ett snitt för att förhindra att fler tester görs. Avsluta sedan med ”fail”. Om det inte finns ett negativt element fungerar `genereramera` som i 11a. (2p)

12. (a) Exempellösning: (1p)

```
squarediff(X,Y,D) :- D is (X-Y)*(X-Y).
```

(b) Exempellösning: (4p)

```
rmsd(L1, L2, RMSD) :-
        msd(L1, L2, MSD), RMSD is sqrt(MSD).

msd([], [], 0).
msd([H1|T1], [H2|T2], MSD) :-
        squarediff(H1, H2, D),
        msd(T1,T2, MSD2),
        MSD is D + MSD2.
```

Poängavdrag görs bl.a. vid följande missar:

- sqrt har inte använts efter “is”.
- Basfallet saknas i rekursivt predikat.
- rmsd har fått extra parametrar.

Syntaxanalys

13. Reglerna kallas *produktioner* och symbolerna kallas *terminaler*. En poäng för vardera ordet.

14. (a) Två förslag: (1p)

    - Det triviala,

      `xyx|xyyyx|xyxyx|xyyxxyyx`

    - Det lite mer ambitiösa,

      `x(y+|y+x+y+)x`

    (b) Förslag:

    `(AC)+|(CA)+|(AG)+|(GA)+|(AT)+|(TA)+|(CG)+|(GC)+|(CT)+|(TC)+|(GT)+|(TG)+`

    Inget poäng om man exempelvis svarat ([ACGT][ACGT])+ eftersom det passar på vilka kombinationer som helst av jämn längd.

    Inte heller `(AC|CA|AG|GA|...|GT|TG)+` duger eftersom det till exempel beskriver `ACCAAAGGTGT`. (1p)

15. Uttrycket består av en alternering av två deluttryck. Det första har 2×2 möjligheter, och det andra har 3 möjligheter. *Men*, de två deluttrycken överlappar och det andra uttrycket har bara ett ord (`xw`) som inte det första har. Alltså finns det fem ord i språket. (1p)

16. En grammatik för S-uttryck där vi använder terminalerna `INT`, `STR`, och `SYM` för att representera S-uttryckens atomer som vi använder. Vi låter terminalerna `LP` och `RP` representera vänster- och höger-parantes, samt punkt med hjälp av `PKT`. (3p)

```
sexp ::= atom
       | pair
       | list
       ;

atom ::= INT | STR | SYM | LP RP;

pair ::= LP sexp PKT sexp RP ;

list ::= LP sexplist RP ;

sexplist ::= (* the empty list *)
           | sexp sexplist
           ;
```

Notationen `(* .. *)` är en kommentar.

Inga avdrag för problem med notationen. Saknade fall och ej fungerande rekursion ska ge -1 vardera.