Omtenta i Programmeringsparadigm 2016-03-15 08.00-11.00

Inga hjälpmedel är tillåtna. Skriv inga svar på blanketten. Varje del är på 20 poäng. För att bli godkänd på en del krävs 12 poäng på den delen. **Du behöver inte skriva de delar där du redan är godkänd på kontrollskrivningen eller ordinarie tentan.**

Del 1: Funktionell programmering

1. Givet koden

```
sub x y = x - y

annat x y = sub y x
```

   (a) Förklara detaljerat vad som händer vid anropet `(annat 2)` dvs där det är möjligt/tillämpbart förklarar du vilka värden som tilldelas till vad, vilka operationer/funktioner som utförs samt vad resultatet av anropet är. *(3p)*

   (b) I funktionen `annat` hårdkodas användningen av den underliggande funktionen `sub`. Skriv en ny funktion `generell_annnat` som gör samma sak, fast där den underliggande funktionen kan anges som argument. T.ex. ska man kunna skriva `generell_annat sub` för att få den ursprungliga funktionen `annat`. *(2p)*

   (c) Hitta på ett nytt, mer beskrivande, namn till funktionen `generell_annat`. Namnet ska reflektera vad funktionen gör. *(1p)*

   (d) Ange typsignaturen för funktionen `generell_annat`. *(2p)*

2. Avståndet mellan två vektorer kan beräknas med t.ex. Euklidiskt avstånd och Manhattan-avstånd. Följande uppgift och kod representerar vektorer (x, y) i det 2-dimensionella planet. Avståndet mellan två vektorer (x_1, y_1) och (x_2, y_2) kan beräknas i Haskell enligt:
   Euklidiskt avstånd: `sqrt((x1 - x2)**2 + (y1 - y2)**2)`
   Manhattan-avstånd: `abs(x1 - x2) + abs(y1 - y2)`

   Givet koden

```
data Vector2D = Vector2D Double Double
              deriving (Eq, Show)

data Norm = Manhattan Vector2D Vector2D |
            Euklidisk Vector2D Vector2D
            deriving (Show, Eq)
```

   (a) Skriv en funktion för att beräkna Mahattan-avstånd mellan två vektorer av typen `Vector2D`. Funktionens typsignatur ska vara: `distance :: Norm -> Double` *(3p)*

   (b) Ge ett exempel på hur man kan anropa funktionen som beräknar Manhattan-avstånd *(1p)*

3. Givet koden

```
adderaPositivaHeltal 0 tal2 = tal2

adderaPositivaHeltal tal1 tal2 =
     adderaPositivaHeltal (tal1 - 1) (tal2 + 1)
```

   (a) Ange de beräknings-anrop som anropet `adderaPositivaHeltal 3 4` ger upphov till. *(2p)*

(b) Motivera tydligt och noggrannt om funktionen är svansrekursiv eller ej. *(2p)*

(c) Varför är rekursion viktigare i det funktionella paradigmet än i det imperativa? *(2p)*

(d) Skriv om den givna koden så att en lista med par returneras som beskriver beräkningsanropen för ett givet anrop. Varje par i listan anger de värden `tal1` respektive `tal2` har i ett givet beräkningsteg. Med andra ord ska man kunna köra den förändrade koden med parametrarna 3 och 4 för att få ut ett svar på uppgift 3(a).

Funktionen ska vara rekursiv och får inte använda listomfattning eller `map`. *(2p)*

Del 2: Logikprogrammering

4. **Kontrollflöde, Negation**

Predikatet `member(X, L)` är sant om och bara om elementet `X` finns med i listan `L`. Predikatet kan definieras så här:

```
member(X, [X|_]).
member(X, [_|T]) :- member(X, T).
```

Utifrån den definitionen, beskriv kontrollflödet, unifieringarna samt slutsvaret vid frågan:

```
?- member(X, [1, 2]), \+ member(X, Y).
```

(6p)

5. **Logisk versus procedurell läsning**

(a) Vad menas med logisk läsning av ett Prolog-program och vad menas med procedurell läsning? *(2p)*

(b) Ge ett exempel på två program som har samma (dvs ekvivalent) logiska läsning, men olika procedurell läsning, där det första programmet fungerar medan det andra inte fungerar. Förklara ditt tankesätt. *(2p)*

6. **Induktiva datatyper, Rekursion**

Betrakta datastrukturen binära träd `T` med data som är lagrad i löven.

```
T ::= empty | N
N ::= leaf(D) | branch(N, N)
```

Data i löven behöver ej vara sorterat, och det är tillåtet för samma data att förekomma i mer än ett löv.

(a) Definiera ett predikat `delEl(-X, -T, +TX)` som är sant om och endast om `TX` är ett träd med samma uppsättning data förutom att det inte ska finnas några förekomster kvar av datat `X`.

Förklara hur du tänkt i din lösning.

OBS: `empty` får inte förekomma i ett icke-tomt träd! *(8p)*

(b) Enligt din lösning, vad blir svaret på frågan:

```
?- delEl(2, branch(leaf(2), branch(leaf(3), leaf(2))), T).
```

(2p)

Del 3: Formella språk och syntaxanalys

7. **Ändliga automater**

   Språket över tecknen "(" och ")" som innehåller alla balanserade parentesuttryck är ett välkänt exempel på ett språk som inte är reguljärt. Men om man begränsar nästlings-djupet på parentesuttrycket (som i labb S1) så blir språket reguljärt.

   Med "djupet" för ett parentesuttryck menar vi det maximala antalet nästlade parentespar. Djupet för "`()`" är 1, och djupet för "`(()(()()))`" är 3.

   Rita en ändlig automat (DFA) som beskriver balanserade parentesuttryck som har djup högst 5. Ange starttillstånd och accepterande tillstånd tydligt. *(4p)*

8. **Grammatiker**

   I den här uppgiften studerar vi balanserade parentesuttryck med olika typer av parenteser: vanliga parenteser "`()`", hakparenteser "`[]`", och måsvingar "`{}`".

   En sträng på dessa sex tecken är balanserad om varje "`(`" matchas av ett "`)`", varje "`[`" matchas av ett "`]`", och varje "`{`" matchas av ett "`}`".

   Exempel på balanserade uttryck är "`{}`", "`{}[]()`", "`([{([{}])}])`", "`{{{}}()}`", och "`{[[(){}]]}`".

   Exempel på icke balanserade uttryck är "`{{}`", "`{{]}`", "`(((`)`]`", "`][`", och "`](`)".

   OBS: Begränsningen från föregående uppgift, att vi bara tittar på begränsat djup, gäller *inte* i denna uppgift.

   (a) Skriv en grammatik för balanserade uttryck med dessa tre parentestyper. *(4p)*

   (b) Är din grammatik LL(1), dvs, kan den parsas med rekursiv medåkning? Motivera ditt svar! *(2p)*

   (c) Välj en sträng på minst 9 tecken och rita ett parse-träd för strängen enligt din grammatik. *(2p)*

   (d) *Bevisa* att språket för balanserade uttryck med dessa tre parentestyper inte är reguljärt. (Du får använda att vanliga balanserade parentesuttryck inte är ett reguljärt språk.) *(2p)*

9. **Egenskaper hos språk-klasser**

   För vart och ett av dessa tre påståenden, ange **tydligt** om det är sant eller falskt, och motivera varför. (1p för rätt svar sant/falskt, 1p för korrekt motivering).

   (a) Om L *inte* är ett reguljärt språk så är språket "alla strängar i L av udda längd" inte heller reguljärt. *(2p)*

   (b) Om L är ett kontextfritt språk så är språket L^* också kontextfritt. *(2p)*

   Språket L^* består av alla strängar som kan skrivas som en följd av strängar från L (på samma sätt som vi i reguljära uttryck kan skriva t.ex. "`([XY]AB)*`" för "en följd av strängar som alla matchar `[XY]AB`").

   (c) Om L kan beskrivas av en ändlig automat (DFA) så kan man skriva en LL(1)-grammatik (dvs en grammatik som kan parsas med rekursiv medåkning) för språket. *(2p)*