Tenta i Programmeringsparadigm 2015-01-16 14.00-17.00

Inga hjälpmedel är tillåtna. Skriv inga svar på blanketten. Bonuspoäng från Inet-labben hösten 2014 kommer automatiskt att tillgodoräknas på den del av tentan där de gör mest nytta. 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.**

Del 1: Funktionell programmering

1. (11p)

   När en funktion anropas utförs ett antal beräkningssteg. Vi har en funktion `f`, som vid anrop med argumenten `funk` och `[1,2,3]` ger följande beräkningssteg.

```
f funk [1,2,3]
           (funk 1) : (f funk [2,3])
           (funk 1) : (funk 2) : (f funk [3])
           (funk 1) : (funk 2) : (funk 3) : (f funk [])
           (funk 1) : (funk 2) : (funk 3) : []
           (funk 1) : (funk 2) : [(funk 3)]
           (funk 1) : [(funk 2), (funk 3)]
           [(funk 1), (funk 2), (funk 3)]
[(funk 1), (funk 2), (funk 3)]
```

   (a) Vilken paradigm (funktionell eller imperativ) har troligen använts för att implementera `f` (med hänsyn till de två paradigmens utmärkande egenskaper)? Motivera ditt svar! *(3p)*

   (b) Förklara hur funktionen `f` skulle implementeras med egenskaper typiska för det andra paradigmet och hur det skulle skilja sig från den första implementationen (med avseende på variabler, anrop, beräkningssteg, etc). *(3p)*

   (c) Funktionen kan implementeras med svansrekursion. Ange de beräkningsteg som erhålls då samma anrop som ovan görs med en svansrekursiv implementation. *(2p)*

   (d) Förklara vad fördelen är med den svansrekursiva lösningen jämfört med den ursprungliga rekursiva lösningen. *(1p)*

   (e) Funktionen skrivs om i Haskell (med bibehållen funktionalitet) så att den använder sig av listomfattning (list comprehension). Hur ser den nya funktionen ut? (Du får här INTE använda fördefinierade funktioner som t.ex. map eller foldr.) *(2p)*

2. (9p)

   Vi har följande Haskell-definitioner:

```
data Tree a =
    Leaf a | Node (Tree a) a (Tree a)
    deriving(Show)

treeFnc qwe (Leaf x) = Leaf (qwe x)
treeFnc qwe (Node t1 x t2) = Node (treeFnc qwe t2) (qwe x) (treeFnc qwe t1)
```

   (a) Hur ser typsignaturen för `treeFnc` ut? *(3p)*

   (b) Förklara i ord vad `treeFnc` gör. *(3p)*

   (c) Vad blir resultatet av anropet

```
treeFnc (+1) (Node (Leaf 1) 2 (Node (Leaf 3) 4 (Leaf 5)))
```

   (3p)

Del 2: Logikprogrammering

3. **Kontrollflöde, Backtracking** (8p)

   Betrakta Prologprogrammet nedan, som genererar permutationer av en lista.

```
permutation([], []).
permutation([E | X], Y) :-
   permutation(X, Y1),
   append(Y2, Y3, Y1),
   append(Y2, [E | Y3], Y).
```

   Predikatet `append(X, Y, Z)` är det inbyggda listhanteringspredikatet som är sant när listan `Z` är lika med konkateneringen av listan `X` med listan `Y`.

   I din lösning på följande uppgifter kan du behandla `append` som en svart låda och ska inte beskriva kontrollflödet inuti `append`. Om du inte råkar veta i vilken ordning `append` genererar lösningar så får du anta någon ordning – så länge ditt svar är korrekt i övrigt ger detta full poäng.

   (a) Beskriv kontrollflödet samt unifieringarna steg för steg vid evalueringen av `permutation([0], Y)`. *(4p)*

   (b) Beskriv kontrollflödet samt unifieringarna steg för steg vid evalueringen av `permutation([1, 2], Y)`.
   Du kan använda och referera till din lösning från uppgift (a). *(4p)*

4. **Unifiering, Rekursion** (5p)

   Skriv ett predikat `count(X, Y, Z)` som är sant om `Z` är lika med antalet förekomster av `X` i listan `Y`. Du får inte använda några inbyggda listhanteringspredikat.

5. **Generera och testa** (7p)

   Ett bräde i spelet tre-i-rad kan representeras som en lista L av längd 9 som innehåller konstanterna 'X', 'O', eller '-' (sista symbolen står för tom ruta).

   Generera-och-testa är en vanlig programstruktur i Prolog för att lösa problem. Vi ska nu använda generera-och-testa-principen för att skapa alla möjliga tre-i-rad-bräden. Villkoret som ska uppfyllas är att det totala antalet X och O i ett bräde skiljer sig med högst 1.

   Du ska nu beskriva hur en sådan lösning kan se ut. Du får gärna använda Prologkod men det är OK att beskriva delar med ord där du inte kommer ihåg syntaxen, så länge din beskrivning är tydlig. Du får använda predikatet `count(X, Y, Z)` från förra uppgiften (oavsett om du löst den eller inte).

   (a) Beskriv den övergripande strukturen på predikatet för att skapa alla bräden. *(1p)*

   (b) Beskriv "generera"-delen av programmet. *(3p)*

   (c) Beskriv "testa"-delen av programmet. *(3p)*

Del 3: Formella språk och syntaxanalys

6. **Allmänt om formella språk och syntaxanalys** (4p)

   (a) Vad är kraftfullast (kan beskriva flest språk), stackautomater (PDA) eller (kontextfria) grammatiker? Eller är de lika kraftfulla? (Bara svar, motivering behövs ej.) *(1p)*

   (b) Hur många ändliga språk finns det? *(1p)*

   (c) Vad är skillnaden/relationen mellan en ändlig automat (DFA) och en stackautomat (PDA)? *(1p)*

   (d) Vad är en *slutsymbol* inom syntaxanalysen? *(1p)*

7. **Reguljära språk och uttryck** (8p)

   Betrakta de två reguljära uttrycken

   $$R_1 = \texttt{p(rog)*p} \qquad\qquad R_2 = \texttt{(prog)*p}$$

   Låt L_1 vara språket som beskrivs av R_1, och L_2 språket som beskrivs av R_2.

   (a) Rita en ändlig automat (DFA) för något av språken L_1 eller L_2. Du får själv välja vilket men måste tydligt ange vilket du valt. Ange också tydligt vad som är start-tillstånd och vad som är accepterande tillstånd. *(2p)*

   Beskriv vart och ett av de tre språken nedan, dels i ord (1p/språk), och dels med ett reguljärt uttryck (1p/språk).

   (b) Språket $L = L_1 \cap L_2$ (snittet av L_1 och L_2). *(2p)*

   (c) Språket $L = L_1 \cup L_2$ (unionen av L_1 och L_2). *(2p)*

   (d) Språket $L = L_1 \setminus (L_1 \cap L_2)$ (L_1 minus L_2). *(2p)*

8. **Kontextfria grammatiker** (8p)

   Vi är intresserade av ett språk för att beskriva riktade grafer. En graf består av en mängd noder, och en mängd kanter, där varje kant går från en nod till någon annan nod.

   Strängarna i språket vi är intresserade av består av en komma-separerad lista med nodbeskrivningar. Varje nodbeskrivning omges av parenteser och innehåller namnet på noden, följt av en kommaseparerad lista (omgiven av måsvingar) på vilka noder kanterna från noden går till. Till exempel skulle det kunna se ut så här:

   ```
   (Nod1,{Nod2,Nod3}),(Nod2,{Nod1}),(Nod3,{})
   ```

   Denna sträng beskriver alltså en graf med tre noder (med namn `Nod1`, `Nod2`, och `Nod3`), där `Nod1` har kanter till de två andra noderna, `Nod2` bara har en kant till `Nod1`, och `Nod3` inte har några utgående kanter alls.

   Vi är inte intresserade av att kontrollera semantiska egenskaper, som huruvida kanterna från en nod leder till noder som faktiskt existerar. Så följande sträng är godkänd:

   ```
   (Nod,{IckeExisterandeNod}),(AnnanNod,{Nod,Nod,Nod})
   ```

   Du kan anta att en lexikal analys har utförts och producerat en indatasekvens med slutsymbolerna LPAREN, RPAREN, LBRACE, RBRACE, COMMA, och NAME (de fem första representear vänsterparentes, högerparentes, vänstermåsvinge, högermåsvinge, respektive kommatecken, och den sista representerar ett nodnamn).

   (a) Skriv en kontextfri grammatik för grafer enligt specifikationen ovan. Ange tydligt vilka icke-slutsymboler som finns, och vad som är startsymbol. *(6p)*

   (b) Är din grammatik från uppgift (a) LL(1)? Motivera ditt svar! *(2p)*