Omtenta i Programmeringsparadigm 2014-03-13 09.00-12.00

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 tenta.**

Prolog

1. **Backtracking** (6 p)

   (a) Betrakta Prologprogrammet nedan:

   ```
   predA(X, X).
   predA(X, Y) :-
     predA(X, Y1),
     Y is Y1+1.
   ```

   Vilka blir det första och andra resultatet på målet `predA(3, Y)`? *(3p)*

   (b) Betrakta följande forsättning på programmet:

   ```
   predB([], []).
   predB([X|Xs], [Y|Ys]) :-
     predA(X, Y),
     predB(Xs, Ys).
   ```

   Vilka blir det första och andra resultatet på målet `predB([1,3,5], L)`? *(3p)*

2. **Snitt** (6p)

   Ge ett litet men meningsfullt exempel på *rött snitt*, dvs ett Prologprogram med snitt där snittet faktiskt behövs för att programmet ska bete sig korrekt.

   Förklara ditt resonemang och varför snittet behövs. Det viktiga är inte att din Prologkod är klanderfri utan att ditt resonemang är tydligt.

3. **Rekursion, Träd** (8p)

   Betrakta binära träd byggda med konstruktorerna '`leaf`' och '`branch(Tree, Tree)`'.

   (a) *Höjden* av ett träd är maximala längden av stigarna från trädets rot till något löv. Skriv ett predikat `height(T, N)` som är sant exakt när höjden av binära trädet `T` är `N`. Du kan anta att det redan finns ett predikat `max(X, Y, Z)` som är sant exakt när `Z` är det större talet av `X` och `Y`. *(4p)*

   (b) Ett binärt träd kallas för *fullständigt* (eng. complete) när alla dess löv är på samma djup. Skriv ett predikat `complete(T)` som är sant exakt när det binära trädet `T` är fullständigt. Du får använda predikatet `height(T, N)` från förra uppgiften (även om du inte löst den). *(4p)*

Haskell

4. (4p)

   (a) Skriv en funktion `lengthEqThree` som kollar en listan har längd 3. *(2p)*
   Exempel som ska ge resultatet `False`:
   `lengthEqThree []`
   `lengthEqThree [1]`
   `lengthEqThree ['e','w']`
   `lengthEqThree [2.3, 5.6, 0.0, 5.6]`
   Exempel som ska ge resultatet `True`:
   `lengthEqThree [51,62,39]`

   (b) Ändra nu `lenghtEqThree` så att den endast tar listor med godtyckliga tal. *(2p)*
   Exempel:
   `lengthEqThree ['e','w']` ska ge ett felmeddelande, övriga exempel ovan är oförändrade.

5. (12p)

   Du ska i denna uppgift skriva funktioner som löser samma problem men på olika sätt. Problemet är: Givet en godtyckligt lång lista med dellistor (en dellista innehåller godtyckligt många heltal) ska innehållet i de dellistor vars längd är tre summeras och returneras, dellistor vars längd inte är tre tas bort.

   Du får i samtliga deluppgifter nedan använda dig av funktionen `lengthEqThree` från (b)-uppgiften ovan.

   Exempel:
   `sumsOfThrees [[]]` ger resultatet `[]`
   `sumsOfThrees [[], [1,2,3,4]]` ger resultatet `[]`
   `sumsOfThrees [[],[1,2,3],[1,2,4],[1,2,3,4]]` ger resultatet `[6,7]`

   (a) Skriv funktionen `sumsOfThrees` rekursiv men **inte** så att den använder sig av den svansrekursiva idén, högre ordningens funktioner eller listomfattning. *(3p)*

   (b) Skriv funktionen `sumsOfThrees` med hjälp av minst de tre funktionerna `map`, `filter`, och `lengthEqThree` (här är det alltså inte valfritt att använda `lengthEqThree` utan ett krav). Använder du andra, egendefinierade, funktioner ska dessa redovisas. *(3p)*

   (c) Skriv funktionen `sumsOfThrees` med hjälp av listomfattning. *(3p)*

   (d) Om problemet löses med hjälp av den svansrekursiva idén, förklara hur det skiljer sig från den rent rekursiva idén. Visa också hur varje anrops/beräkningssteg ser ut för `sumsOfThrees [[1,2,3], [4,5], [6,7,8]]` när den svansrekursiva idén tillämpas. *(3p)*

6. (4p)

   Man vill skriva ett program där man jobbar med vektorer av tre olika slag; en-, två- respektive tre-dimensionella. Exempel på vektorer av respektive slag:
   1D: {2.3}, 2D: {5.3, 6.7}, 3D: {5.0, 4.67, 1.2}

   Man vill att datatypen ska vara definierad på ett sådant sätt att man kan jämföra om två vektorer är komponentvis lika med "=="-operatorn.

   Man har börjat med koden:

   ```
   data Vector =
   ```

   Skriv färdigt definitionen av datatypen `Vector` så att man utifrån den kan arbeta med de tre olika vektorslagen som tänkt.

Syntax

7. (4p)

   Följande mening saknar tre delar.

   "En/ett .... **(A)** .... beskriver en/ett .... **(B)** .... som kan kännas igen av en/ett .... **(C)** .... "

   Använd följande termer för att fullborda meningen på två olika sätt så att meningen blir korrekt.

   (1) Kontextfri grammatik
   (2) Kontextfritt språk
   (3) Reguljärt språk
   (4) Reguljärt uttryck
   (5) Stackautomat (PDA)
   (6) Ändlig automat (DFA)

   Varje term ska användas exakt en gång. Ditt svar skulle alltså kunna se ut så här (förutom att det här inte är rätt svar):
   "Mening 1: A-5 B-3 C-4
   Mening 2: A-2 B-1 C-6"

8. (7p) I det här problemet kommer vi att arbeta med ett språk för att göra vägbeskrivningar. En vägbeskrivning är en sekvens av en eller flera instruktioner. I språket finns instruktionerna `V x` för att svänga `x` grader åt vänster, `H x` för att svänga `x` grader åt höger, `F x` för att gå `x` meter framåt, och slutligen `R x ( ... )` för att upprepa instruktions-sekvensen "`...`" `x` gånger. `R`-instruktioner kan vara nästlade (se sista exemplet nedan).

   Exempel:
   `F 10 H 45 F 10` säger "gå framåt 10 meter, sväng 45 grader åt höger, gå framåt 10 meter"
   `R 90 (V 1)` upprepar "sväng vänster en grad" 90 gånger, och är ekvivalent med `V 90`
   `R 4 (F 1 V 90)` upprepar "går framåt en meter och sväng 90 grader vänster" fyra gånger (man har då gått i en kvadrat och är tillbaka där man började, i samma riktning).
   `R 42 (F 10 R 4 (F 1 V 90) )` upprepar "gå 10 meter framåt och gå sedan runt i en liten kvadrat" 42 gånger.

   (a) Skriv en kontextfri grammatik för vägbeskrivningsspråket med slutsymbolerna `V`, `H`, `F`, `R`, `VP` (vänsterparentes), `HP` (högerparentes), och `Num` (tal). *(5p)*
   (b) Rita ett syntaxträd för vägbeskrivningen `R 4 (F 1 V 90)` enligt din grammatik ovan. *(2p)*

9. (9 p)

   I det här problemet kommer vi att arbeta med att känna igen strängar som är heltal. Vi är dock petiga och vill inte att strängarna ska ha någon redundans i form av inledande nollor eller negation av talet 0.

   Exempel på strängar som är godkända
   `-9876543210`, `0`, `-2014`, `10000000000000000000000000000000000000000000000000`

   Exempel på strängar som *inte* är godkända
   `78z`, `007`, `-007`, `0-7`, `-0`, `753.1`, `--2014`

   (a) Skriv ett reguljärt uttryck för heltal enligt dessa begränsningar. *(3p)*
   (b) Rita en ändlig automat (DFA) som känner igen heltal enligt dessa begränsningar. Ange tydligt vad som är starttillstånd och vad som är accepterande tillstånd. *(3p)*
   (c) Skriv en kontextfri grammatik för heltal enligt dessa begränsningar. *(3p)*