Tenta i Programmeringsparadigm 2018-01-10 14.00-17.00

Inga hjälpmedel är tillåtna. Skriv inga svar på blanketten. Bonuspoäng från Inet-labben hösten 2017 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.** Lycka till önskar Dilian, Marcus och Per.

Del 1: Funktionell programmering

1. I det imperativa paradigmet kan du återanvända ett variabelnamn godtyckligt antal gånger, men i en Haskellfunktion kan du bara binda ett uttryck till ett namn en gång. (1) Namnge den funktionella principen för detta (engelska går bra) och (2) förklara vad vi har för nytta av den. *(2p)*

2. Typsignaturen för elem ser ut såhär:

   ```
   elem :: (Eq a, Foldable t) => a -> t a -> Bool
   ```

   (a) Vad är a och t exempel på i typsignaturen ovan? *(1p)*

   (b) Vad kallas principen att en funktion kan anropas med olika typer och varför vill vi kunna göra det? *(2p)*

3. (a) Skriv en Haskellfunktion som representerar en oändlig lista med jämna heltal utan att upprepa sig. Kom ihåg typsignaturen. *(2p)*

   (b) (1) Visa **med kod** hur du kan anropa funktionen i 3a och plocka ut delar av listan utan att fastna i en oändlig loop, (2) namnge och (3) förklara den princip för funktionell programmering som Haskell använder vid funktionsanrop som gör ditt anrop möjligt. *(3p)*

4. En lista v innehåller n heltal, $n \geq 1$. Funktionen s är definierad för varje element i listan enligt följande:
   $s(0) = v(0)$
   $s(i) = s(i-1)^2 + v(i), 0 < i < n, i \in \mathbb{Z}$
   Implementera en Haskellfunktion som beräknar $s(n-1)$ där n är antalet element i listan (observera 0-indexeringen). Du får inte använda rekursion. Din lösning måste istället använda högre ordningens funktioner och (minst) en funktion definierad med lambdanotation. *(5p)*

5. Skriv en Haskellfunktion som tar en lista v med strängar och filtrerar ut den (1) den första förekomsten av strängen "HALT" samt (2) alla förekomster av strängen "SKIP" som har lägre index än den första förekomsten av "HALT". Om det inte finns någon "HALT" så filtreras alla "SKIP" ut ur listan.

   Exempelkörningar:

   ```
   Prelude> skip ["DO", "NOT", "LECTURE", "SKIP", "HALT", "ME", "ABOUT", "THE LAW"]
   ["DO","NOT","LECTURE","ME","ABOUT","THE LAW"]
   Prelude> skip ["SKIP", "HALT", "SKIP", "HALT", "MELTDOWN", "SPECTRE"]
   ["SKIP","HALT", "MELTDOWN", "SPECTRE"]
   Prelude>
   ```

   Du får använda hjälpfunktioner och inbyggda funktioner. *(5p)*

Del 2: Logikprogrammering

6. **Kontrollflöde** *(6p)*

   Följande Prologpredikat kollar om ett icke-negativt heltal är jämt eller inte:

   ```
   even(0).
   even(X) :- Y is X-1, \+ even(Y).
   ```

   Givet denna deklaration, beskriv *kontrollflödet* med (1) alla *regelinstanser*, (2) alla *unifieringar*, och (3) alla eventuella *backtrackningar* under exekveringen av frågan:

   ```
   ?- even(2).
   ```

   OBS: Presentera kontrollflödet i stilen som vi har använt på föreläsningarna. Svar baserade på lådmodellen eller trace kommer inte att ge några poäng.

7. **Induktiva datatyper: Listor** *(6p)*

   Som bekant är det inbyggda Prologpredikatet över listor `append(X, Y, Z)` sant när listan `Z` är konkateneringen av listan `X` med listan `Y`.

   Använd predikatet för att implementera ett nytt Prologpredikat `appendAll(X, YY, ZZ)`, där `X` är en lista medan `YY` och `ZZ` är listor av listor, som är sant när `ZZ` innehåller konkateneringen av `X` med varje lista i `YY`, i samma ordning som i `YY`. T.ex. ska frågan:

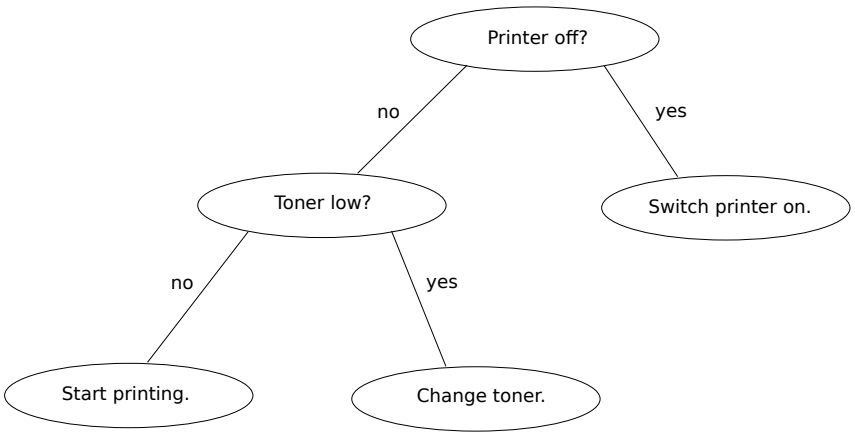
   ```
   ?- appendAll([a, b], [[c, d], [e, f]], ZZ).
   ```

   ge svaret:

   ```
   ZZ = [[a, b, c, d], [a, b, e, f]] .
   ```

8. **Induktiva datatyper: Binära träd med data**

   Binära träd med data är in icke-inbyggd datatyp som byggs med konstruktörerna `leaf(Data)` och `branch(Data, Tree, Tree)`. Vi ska använda oss av datatypen för att representera så kallade *beslutsträd*.



Figur 1: Ett enkelt beslutsträd.

Beslutsträd är binära träd som innehåller frågor vid sina interna noder och anvisningar vid sina löv. Man börjar i roten. Om den är en intern nod (`branch`), så ställer man den

associerade frågan till användaren (genom att skriva ut den på konsolen) och man inväntar ett svar (från terminalen). Om svaret är `no`, fortsätter man rekursivt med vänstra barnet, och om svaret är `yes` med högra barnet. T.ex. kan interaktiva felsökningsguider ges som beslutsträd.

(a) Vilken Prologterm representerar beslutsträdet på bilden i Figur 1? Termen måste vara av typen binärt träd med data. Frågor och anvisningar skall representeras som Prologsträngar, som t.ex. `'Printer off?'`. *(2p)*

(b) Skriv ett Prologpredikat `process(BT)`, som givet beslutsträdet `BT`, bearbetar det interaktivt enligt beskrivningen ovan. *(6p)*

I lösningen får man använda det inbyggda Prologpredikatet `read(t)` som inväntar input från terminalen (som avslutas med punkt) och unifierar den med termen `t`, och det inbyggda Prologpredikatet `write(t)` som skriver ut termen `t` till konsolen.

Del 3: Formella språk och syntaxanalys

9. **Reguljära språk**

Denna uppgift handlar om ett språk för bingo-resultat.

En sträng är ett bingo-resultat om den är en icke-tom sekvens med tal mellan 1 och 75, separerade av kommatecken. T.ex. är "`5,12,75,23`" och "`1`" bingo-resultat, men inte "`76,12`", "`5,0`" eller tomma strängen. Vi tillåter inte heller inledande nollor i tal, så "`07`" är inte ett bingo-resultat. Vi tillåter dock upprepade tal, så "`7,7,7,7,7`" är ett giltigt bingo-resultat. Notera att detta är strängar över tecknen '`0`'-'`9`' samt komma-tecken.

En bingo-bricka är en 5×5-matris fylld med tal mellan 1 och 75, t.ex.

$$\begin{pmatrix} 7 & 19 & 37 & 46 & 70 \\ 3 & 20 & 31 & 50 & 61 \\ 15 & 21 & 40 & 51 & 66 \\ 8 & 22 & 42 & 47 & 71 \\ 1 & 23 & 45 & 60 & 75 \end{pmatrix}$$

En bingo-bricka har fått bingo i ett bingo-resultat om alla fem tal från någon av kolumerna, raderna, eller diagonalerna i brickan förekommer i resultatet. T.ex. har brickan ovan bingo i resultaten "`19,20,21,22,23`", "`40,20,15,66,51,75,21`" och "`70,1,1,1,50,22,40`", men inte i "`3,20,31,50,51,47,60`".

För vart och ett av följande språk, **ange tydligt** om det är reguljärt eller inte (1p per uppgift), och motivera sedan varför (2p per uppgift, formellt bevis krävs inte).

(a) Språket med alla strängar som är bingo-resultat där bingo-brickan som visas ovan har bingo. *(3p)*

(b) Här är en till bingo-bricka:

$$\begin{pmatrix} 1 & 15 & 31 & 45 & 61 \\ 2 & 16 & 32 & 46 & 62 \\ 3 & 17 & 33 & 47 & 63 \\ 4 & 18 & 34 & 48 & 64 \\ 5 & 19 & 35 & 49 & 65 \end{pmatrix}$$

Vi vill nu ha språket med alla strängar som är bingo-resultat där den första bingo-brickan ovan har bingo men där den här nya brickan *inte* har bingo. *(3p)*

(c) Vi börjar ångra att vi tillåter samma tal att upprepas flera gånger i ett bingo-resultat. I den här uppgiften vill vi ha alla bingo-resultat där inget tal upprepas. (Oavsett om någon av de båda bingo-brickorna ovan har bingo eller inte.)

(3p)

10. **Grammatiker**

I denna uppgift får du hitta på ett språk efter egen fantasi (eller använda dig av något av dina favoritspråk). Du ska sedan göra några olika uppgifter med språket. Läs igenom deluppgifterna och kraven nedan innan du väljer språk så att du inte råkar välja ett språk för vilket några av deluppgifterna saknar lösning.

Ditt språk måste uppfylla följande krav:

- Språket ska vara definierat över tecknen '`a`'-'`z`', inga andra.
- Både språket och dess komplement måste vara oändliga.

(a) Beskriv **tydligt** i ord vilket språk du har valt. För att få poäng behöver ditt språk kunna användas för att lösa minst två av de andra deluppgifterna. *(1p)*

(b) Ge exempel på två strängar av längd minst 5 som tillhör språket, och två strängar av längd minst 5 som inte tillhör språket. *(2p)*

De strängar du ger som inte tillhör språket får bara innehålla tecken som förekommer i någon sträng i språket – om du t.ex. ger en sträng som innehåller '`z`' som exempel så måste det ändå finnas andra strängar i språket som innehåller '`z`'.

(c) Konstruera en tvetydig grammatik för språket. Du får inte använda utökad BNF (EBNF-notation). *(2p)*

(d) Visa att din grammatik från (c) är tvetydig genom att ge en sträng med två olika syntaxträd. *(3p)*

(e) Konstruera en grammatik för språket som går att parsa med rekursiv medåkning. Du får inte använda utökad BNF (EBNF-notation). *(3p)*
