Omtenta i Programmeringsparadigm 2017-04-11 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. (a) Skriv en Haskellfunktion som tar en Int som indata och returnerar en sträng med den binära representationen av denna Int. Rekursion måste vara en central del av din lösning. Du får inte använda inbyggda funktioner som direkt löser problemet, till exempel `intAtBase`. Du får använda hjälpfunktioner. Kom ihåg typsignaturen. Du får anta att Int:en är ickenegativ. *(5p)*

   (b) Lista anropen till din(a) funktion(er) då huvudfunktionen i föregående uppgift anropas med Int:en 21. *(3p)*

2. Studera följande kod:

```
listCreator::Int->Int->Int->[Int]
listCreator limit power n =
    (filter (<limit) . map (^power)) [1..n]
```

   (a) Skriv resultatet då den anropas med `listCreator 23 2 10` *(2p)*

   (b) Skriv om listCreator med en listomfattning. Du får inte använda rekursion eller hjälpfunktioner. *(3p)*

3. Typsignaturen för metoden sum är `sum :: (Num a, Foldable t) => t a -> a`. Vilket begrepp beskriver bäst "`a`" i detta exempel? *(2p)*

4. Skriv en funktion som beräknar summan av de första n udda talen. En funktion definierad med lambdanotation ska vara central i din lösning. Du får inte använda rekursion eller hjälpfunktioner. Du får inte använda den inyggda metoden sum. *(5p)*

Del 2: Logikprogrammering

4. **Kontrollflöde**

   På förra tentan handlade en uppgift om att beräkna valresultat från en lista av röster, där varje röst representeras som den kandidat som rösten har gått till, och varje kandidat representeras som en atom (t.ex. `[adam, eva, nisse, eva]`).

   Första del-uppgiften gick ut på att implementera ett predikat `candVotes(+C, +Vs, -N)` som skulle vara sant om och endast om `N` är antalet röster i listan `Vs` som har gått till kandidaten `C`. Ett (inkorrekt!) försök till lösning skulle kunna vara:

```
candVotes(_, [], 0).
candVotes(C, [V | Vs], N) :-
   candVotes(C, Vs, M),
   C = V,
   N is M+1.
```

   Beskriv kontrollflödet samt unifieringarna steg för steg vid evalueringen av följande mål. *(7p)*

```
candVotes(eva, [adam, eva], X).
```

5. **Generera och testa**

   Beskriv kortfattat ansatsen "generera-och-testa" för att lösa problem med logisk programmering. Hur ser den generella strukturen ut i ett Prolog-program som följer ansatsen? *(3p)*

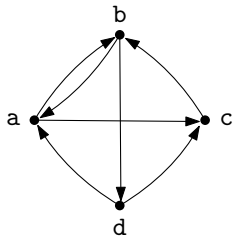
6. **Listor**

   En riktad graf består av en mängd noder och en mängd riktade bågar mellan noderna. En väg i en graf är en nodsekvens där en nod `a` i sekvensen följs av en nod `b` bara om det finns en båge i grafen från `a` till `b`. En *Hamiltonsk stig* i en graf är en väg som innehåller varje nod exakt en gång.

   (a) Föreslå en generell representation av grafer och vägar som termer och listor av termer i Prolog.

   OBS en graf *ska* alltså representeras med en enda term, och *inte* som en samling fakta som i labb L2. *(3p)*

   (b) Ange hur följande graf representeras i din representation. *(2p)*



   (c) Följ "generera-och-testa"-ansatsen för att implementera ett predikat

   `hamiltonian(+Graph, ?Path)`

   som är sant om och endast om `Path` är en Hamiltonsk stig i den givna grafen `Graph`. Predikatet ska kunna användas för att generera alla Hamiltonska stigar i `Graph` genom att skicka in en oinstansierad `Path`-variabel.

   Dela upp implementationen tydligt i en generera- och en testa-del. Förklara din lösning. Du får använda alla predikat som vi har betraktat i kursen med hänvisning till vad de gör (och inte hur).

   Exempel: grafen i (b) ovan har exakt sex Hamiltonska stigar:
   `a` → `b` → `d` → `c`
   `a` → `c` → `b` → `d`
   `b` → `d` → `a` → `c`
   `c` → `b` → `d` → `a`
   `d` → `a` → `c` → `b`
   `d` → `c` → `b` → `a`

   När representationen av denna graf skickas in som `Graph`, ska första svaret unifiera `Path` med representationen av någon av dessa vägar, nästa svar en annan av vägarna, och så vidare. Varje Hamiltonsk stig ska genereras exakt en gång, så totalt ska sex svar genereras för denna exempel-graf. *(5p)*

Del 3: Formella språk och syntaxanalys

Alla uppgifter på denna tenta-del kretsar kring ett språk för att beskriva polynom, t.ex. x^2+y^2+1 eller $5-3\cdot x\cdot y+7\cdot z^{42}\cdot x$. Vi väljer att representera polynom som text-strängar enligt följande regler:

- Ett *polynom* är en följd bestående av en eller flera termer.
- En *term* har ett inledande tecken `+` eller `-`, och sedan en icke-tom följd av faktorer, separerade av `*`. Faktorer i en term är antingen koefficienter eller variabel-potenser.
- En *koefficient* är ett icke-negativt heltal. Det får finnas högst en koefficient i en term, och den måste komma först.
- En *variabel-potens* är ett variabel-namn följt av `^e` där exponenten `e` är ett icke-negativt heltal. Om exponenten är 1 kan exponentdelen skippas, d.v.s. `x^1` kan skrivas som `x`.
- En *variabel* är något av tecknen `a`–`z`.
- Ett icke-negativt heltal är en icke-tom sträng av tecknen `0`–`9` (vi tillåter alltså t.ex. "`0002`").

Exempel på *giltiga* termer är:

```
+05
-000*x^3*y*w*x^1*a^0
-z^2
```

Exempel på *ogiltiga* termer är:

```
-x*5            (koefficienten inte först)
+5*5            (mer än en koefficient)
-3*xy           (ingen * mellan x och y)
z^2             (inget inledande + eller -)
```

Exempel på giltiga polynom:

```
-1-x^2-y^2+42*x*x*x*a*z^42*x+1-0
+x^1
```

Exempel på ogiltigt polynom:

```
5-x             (första termen "5" ogiltig)
```

7. **Grammatiker**

   Skriv en kontextfri grammatik för *termer* enligt de givna reglerna. (OBS bara termer, inte hela polynom.)

   I nästa uppgift ska grammatiken parsas med rekursiv medåkning. Om du skriver en korrekt grammatik som inte kan parsas med rekursiv medåkning får du fortfarande full poäng på den här uppgiften, men du kommer då inte kunna få full poäng på nästa uppgift. *(5p)*

8. **Rekursiv medåkning**

   Vi ska nu parsa termer genom att applicera rekursiv medåkning på din grammatik från förra uppgiften.

   Förklara hur vi ska gå till väga för att göra detta *och* illustrera med pseudokod (du behöver inte skriva fullständig pseudokod för hela parsern, det räcker att demonstrera ett par nyckelfunktioner).

   Om din grammatik har delar som inte går att parsa med rekursiv medåkning kan du fortfarande få upp till 5 poäng på denna uppgift. Du ska då fortfarande förklara hur vi skulle gå till väga för att konstruera parsern och illustrera med pseudokod, men dessutom explicit peka ut vilka delar av grammatiken som inte går att hantera med rekursiv medåkning (och förklara varför så är fallet). *(6p)*

9. **Reguljära uttryck / automater**

   Språket för polynom som definierats ovan är i själva verket ett reguljärt språk. Visa detta, genom att antingen konstruera ett reguljärt uttryck eller en ändlig automat (DFA) för att känna igen polynom enligt de givna reglerna. *(5p)*

   Tänk på att t.ex. `*` och `+` har en speciell betydelse i reguljära uttryck. För att i ett reguljärt uttryck matcha tecknet asterisk (eller plus-tecken) skriver vi `\*` (eller `\+`).

10. **Egenskaper hos reguljära språk**

    I polynom-språket vi definierat tillåts upprepade variabler i en term, och samma term tillåts förekomma flera gånger.

    (a) Antag att vi vill förbjuda termer där samma variabel förekommer mer än en gång, t.ex. `-x*y*x^3*z` (`x` förekommer två gånger).

    Är polynom-språket fortfarande reguljärt med denna restriktion? Motivera ditt svar. *(2p)*

    (b) Antag att vi vill förbjuda termer som använder samma exakt samma variabel-potenser att förekomma mer än en gång i ett polynom, t.ex.:

    ```
    +x^2-3*x^2      ("x^2" förekommer två gånger)
    ```

    Däremot tillåter vi fortfarande polynom som har två termer som använder samma variabel-potenser rent matematiskt, men där de formaterats annorlunda. T.ex. tillåter vi fortfarande

    ```
    +x*y^2+y^2*x+y*y*x+y*x*y
    -y+y^1
    ```

    Observera här att begränsningen från (a) inte gäller i denna uppgift, variabler tillåts upprepas i en term.

    Är polynom-språket fortfarande reguljärt med denna restriktion? Motivera ditt svar. *(2p)*
