Tenta i Programmeringsparadigm 2018-04-04 08.00-11.00

Inga hjälpmedel är tillåtna. Skriv inga svar på blanketten. Bonuspoäng gäller inte på omtentan. 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 tidigare tentor.** Lycka till önskar Dilian, Marcus och Per.

Del 1: Funktionell programmering

1. En sträng innehåller vänsterparenteser, högerparenteser och andra tecken. Din uppgift är att se till att det inte finns några omatchade parenteser. En algoritm för att avgöra det är att gå igenom strängen från vänster och räkna vänster- samt högerparenteser. Om du efter ett något antal tecken i strängen har sett fler högerparenteser än vänsterparenteser så finns det minst en omatchad parentes. Om du har gått igenom hela strängen utan att ha sett lika många vänster- som högerparenteser så är finns det minst en omatchad parentes. Implementera denna algoritm i en Haskellfunktion och använd svansrekursion centralt i din lösning. Funktionen ska returnera `False` om det finns minst en omatchad parentes och `True` annars. Du får använda hjälpfunktioner. Alla funktioner du definierar behöver ha korrekta typsignaturer. *(5p)*

2. Förklara med ett exempel vad **currying** är i Haskell. *(2p)*

3. En enhetsmatris (som också kallas identitetsmatris efter engelskans identity matrix) är en kvadratisk matris med ettor och nollor. Huvuddiagonalen (dvs den diagonal som sträcker sig från matrixens övre vänstra hörn till matrisens nedre högra hörn) består av ettor och resten av matrisen är nollor. Din uppgift är att skriva en haskellfunktion `identity` som tar ett heltal n och skapar en enhetsmatris med höjd och bredd n element. Du får använda hjälpfunktioner. För att få några poäng behöver du använda minst en listomfattning som ska vara central för din lösning. Kom ihåg typsignatur(er).
   Exempel på anrop:

   ```
   *Main> identity 0
   []
   *Main> identity 1
   [[1]]
   *Main> identity 2
   [[1,0],[0,1]]
   *Main> identity 3
   [[1,0,0],[0,1,0],[0,0,1]]
   *Main>
   ```

   (5p)

4. Nedanstående kod är tänkt att räkna ut antalet 1:or i binärrepresentationen för ett heltal. Tyvärr har koden ett fel. Beskriv felet med kursens terminologi och visa två olika sätt att lösa det, dels med pattern matching (PM) och guards (G). Markera med PM och G vilken av lösningarna som använder vilken metod.

   ```
   popcount n = n `mod` 2 + popcount (n `div` 2)
   ```

   (3p)

5. Skriv en funktion oneTimePad som tar två strängar som indata, klartext och nyckel för att sedan producera en kryptotext. Kryptotexten ska vara lika lång som klartexten och för varje bokstav så ska den krypterade bokstaven följa följande algoritm (som är skriven i pseudokod och detaljerna heter oftast inte riktigt samma sak i Haskell):

```
crypto[i] = char((ord(plaintext[i]) + ord(key[i])) % 128)
```

Funktionen ska implementeras med högre ordningens funktioner och ett lambdauttryck på central position. Rekursiva lösningar godkänns inte. Ange typsignaturer på alla funktioner som du definierar. Du får använda hjälpfunktioner. *(5p)*

Del 2: Logikprogrammering

6. **Kontrollflöde** *(8p)*

   Binära träd (utan data) definieras med grammatiken:

   ```
   <Tree>  ::=  leaf  |  branch(<Tree>, <Tree>)
   ```

   Prologpredikatet `height(T, N)` är sant när höjden på binära trädet `T` är `N`:

   ```
   max(X, Y, X) :- Y<X.
   max(X, Y, Y).

   height(leaf, 0).
   height(branch(TL, TR), N) :-
     height(TL, NL),
     height(TR, NR),
     max(NL, NR, M),
     N is M+1.
   ```

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

   ```
   ?- height(T, 1).
   ```

   OBS: Presentera kontrollflödet i stilen som vi har använt på föreläsningarna. Svar baserade på lådmodellen eller `trace` ger inte godkänt.

7. **Induktiva datatyper: Logiska formler (Syntaxträd)** *(12p)*

   Vi betraktar enkla *propositionella logiska formler*, som representeras med Prologtermer enligt grammatiken:

   ```
   <Form> ::= pvar(<PVar>) | neg(<Form>) | and(<Form>, <Form>) | or(<Form>, <Form>)
   <PVar> ::= p | q | r
   ```

   Till exempel, logiska formeln $\neg((p \wedge q) \vee \neg p)$ representeras med Prologtermen:

   ```
   neg(or(and(pvar(p), pvar(q)), neg(pvar(p))))
   ```

   En logisk formel är skriven i *positiv form* om negation *endast* förekommer på propositionella variabler. Med användning av logiska ekvivalenserna:

   $$\begin{array}{rcl} \neg\neg\phi & \Leftrightarrow & \phi \\ \neg(\phi \vee \psi) & \Leftrightarrow & \neg\phi \wedge \neg\psi \\ \neg(\phi \wedge \psi) & \Leftrightarrow & \neg\phi \vee \neg\psi \end{array}$$

   så kan man omföra varje propositionell logisk formel till positiv form. Till exempel kan formeln $\neg((p \wedge q) \vee \neg p)$ skrivas om till den ekvivalenta formeln $(\neg p \vee \neg q) \wedge p$ som är i positiv form och representeras med Prologtermen:

```
and(or(neg(pvar(p)), neg(pvar(q))), pvar(p))
```

Skriv ett Prologpredikat `convert(F, FC)` som är sant om och bara om den logiska formeln `FC` är den logiska formeln `F` överförd till positiv form enligt reglerna ovan.

Del 3: Formella språk och syntaxanalys

8. I den här uppgiften behandlar vi strängar som anger temperaturer i grader Celsius. Den minsta möjliga temperaturen, den absoluta nollpunkten, är -273.15 grader Celcius. Den största möjliga temperaturen, enligt rådande fysikaliska modeller, är Planck-temperaturen som är ca $1.42 \cdot 10^{32}$ grader Celcius.

   För att göra uppgiften lite enklare begränsar vi oss till heltalstemperaturer (så minimumtemperaturen blir -273 grader istället). Vidare definierar vi max-temperaturen till exakt 10^{32} grader.

   Vi tillåter inte ledande nollor, dvs "`023`" och "`00`" är inte giltiga temperaturer, men temperaturen "`0`" grader är såklart tillåten (och tomma strängen är inte en giltig temperatur).

   Exempel på giltiga temperaturer:

   ```
   -273
   0
   4711
   100000000000000000000000000000000
   ```

   Exempel på ogiltiga temperaturer:

   ```
   -274
   -0
   4711.5
   10^32
   100000000000000000000000000000001
   ```

   (a) Skriv ett reguljärt uttryck för temperaturer enligt dessa regler. Uttrycket får vara högst 1000 tecken långt (men du borde inte behöva i närheten av så många tecken). Du får vid behov använda syntaktiskt socker som t.ex. "`[a-z]`", "`x?`", "`y+`", och "`z{5,10}`". *(4p)*

   (b) Konstruera en ändlig automat (DFA) för temperaturer enligt dessa regler. Automaten får ha högst 1000 tillstånd (men du borde inte behöva i närheten av så många). Om din automat blir stor men följer ett tydligt mönster/system räcker det att du tydligt förklarar detta mönster (du behöver alltså inte rita ut alla tillstånd och övergångar i din automat så länge du tydligt förklarar vilka tillstånd och övergångar som finns). *(6p)*

   (c) Ponera att vi ändrar språket och tillåter decimaler, godtyckligt många. Är det fortfarande reguljärt? Motivera ditt svar. *(3p)*

9. *Peano-aritmetiken* är en axiomatisering av aritmetik med naturliga tal, introducerad av den italienske matematikern Giuseppe Peano i slutet av 1800-talet.

   De *enda* två grund-elementen i Peano-aritmetiken är "`0`" (som, föga förvånande, helt enkelt betyder talet 0), och *efterföljare* "$S(n)$" (som betyder nästa tal efter n, där n är ett tal). T.ex. finns alltså inte talet 3 som element utan måste konstrueras som "`S(S(S(0)))`".

   (a) Skriv en grammatik för tal i Peano-aritmetiken. För att få full poäng ska din grammatik vara så enkel som möjligt, och gå att parsa med rekursiv medåkning. *(4p)*

   (b) Är språket med tal i Peano-aritmetiken reguljärt? Motivera ditt svar. *(3p)*
