Göteborgs Universitet och Chalmers Tekniska Högskola 2000–01–01
Datavetenskap INN11

Tentamen i Beräkningsmodeller

Lördagen den 24 Februari 2001, kl 8.45 – 13.45 i VV11
Ansvarig lärare: Bengt Nordström, tel 55 45 25
Tillåtna hjälpmedel: Inga.
Börja varje uppgift på nytt blad. Skriv endast på en sida av papperet. Varje svar skall motiveras! Den här skriftliga tentamen utgör en del (75 %) av den totala examinationen, den andra delen (dvs. 25 %) består av de inlämningsuppgifter som har delats ut under kursens gång. För årets och förra årets elever gäller alltså att summan av poängen från inlämningsuppgifterna och den skriftliga tentan skall vara minst 100 för att få godkänt på kursen. Examensvisning kommer att äga rum fredagen den 9 Mars kl 11.00 i Bengt Nordströms tjänsterum. Lösningar till den här tentan kommer att finnas tillgänglig från kursens hemsida.

1. Ge ett exempel på ett λ-uttryck som terminerar vid normal evauleringsordning men inte vid applikativ evalueringsordning. Motivera! (20)

   Svar: Se läroboken sid 163.

2. Visa hur man kan representera elementen i mängden **Bool** i lambda-kalkyl respektive χ! Visa också hur man kan representera `if`-funktionen samt att den har egenskapen att

   $$\mathtt{if}\ \ulcorner\mathtt{true}\urcorner\ \mathtt{d}\ \mathtt{e} = \mathtt{d}$$

   (10)

   Svar: I lambda-kalkyl:

   $$\begin{aligned}\ulcorner\mathtt{true}\urcorner &=_{\mathrm{def}} \lambda\mathtt{t}.\lambda\mathtt{f}.\mathtt{t}\\ \ulcorner\mathtt{false}\urcorner &=_{\mathrm{def}} \lambda\mathtt{t}.\lambda\mathtt{f}.\mathtt{f}\\ \ulcorner\mathtt{if}\urcorner &=_{\mathrm{def}} \lambda\mathtt{b}.\lambda\mathtt{d}.\lambda\mathtt{e}.\mathtt{b}\ \mathtt{d}\ \mathtt{e}\end{aligned}$$

   I språket χ:

   $$\begin{aligned}\ulcorner\mathtt{true}\urcorner &=_{\mathrm{def}} \mathtt{true}\\ \ulcorner\mathtt{false}\urcorner &=_{\mathrm{def}} \mathtt{false}\\ \ulcorner\mathtt{if}\urcorner &=_{\mathrm{def}} \lambda\mathbf{x}.\lambda\mathbf{y}.\lambda\mathbf{z}.\mathbf{case}\ \mathtt{x}\ \mathbf{of}\ \{\mathtt{true} : \mathtt{y}, \mathtt{false} : \mathtt{z}\}\end{aligned}$$

   Några enkla reduktioner visar att `if`-funktionen uppfyller den efterfrågade likheten (se läroboken sid 173 om du är tveksam).

3. Varför är PRF (mängden av de primitivt rekursiva funktionerna) en dålig modell för allmänt beräkningsbara funktioner? (20)

   Svar: Alla primitivt rekursiva funktioner terminerar, därför kan man inte använda PRF om man vill uttrycka en beräkningsbar funktion som ej terminerar. Ett annat svar är att det finns terminerande beräkningsbara funktioner (t.ex. Ackermanns funktion) som ej kan uttryckas i PRF. Ett tredje svar är att man med ett enkelt diagonaliseringsargument kan konstruera en funktion som är intuitivt beräkningsbar men ej med i PRF, en sådan konstruktion finns i föreläsningsanteckningarna på hemsidan.

4. Enligt läroboken är en icketom mängd A uppräkningsbar om det finns en total surjektiv funktion $f \in \mathbf{N} \to A$.

   (a) Är det väsentligt att funktionen skall vara total? (15)

   (b) Är det väsentligt att funktionen skall vara surjektiv? (15)

   Om svaret är ja, skall du motivera det genom att visa att de reella talen skulle vara uppräkneliga om kravet inte finns med i definitionen. Om svaret är nej, skall du visa hur man givet en funktion som inte uppfyller kravet kan konstruera en funktion med kravet uppfyllt.

   Svar: Det är inte väsentligt att funktionen är total, om vi har en icke-total surjektiv funktion $f \in \mathbf{N} \to A$ så kan vi ju alltid konstruera en total funktion g genom:

   $$g(x) = \begin{cases} f(x) & \text{om } f(x) \text{ är definierad,} \\ a & \text{för övrigt} \end{cases} \qquad (1)$$

   där a är ett godtyckligt element i A.

   Däremot är det viktigt att funktionen är surjektiv. Annars skulle ju identitetsfunktionen räkna upp de reella talen.

5. Bevisa att man i Haskell (eller något annat typat funktionellt språk) inte kan skriva en funktion `halt :: (Nat -> Nat) -> Nat -> Bool` som är sådan att `(halt f i)` evaluerar till `true` om `(f i)` terminerar och annars evaluerar till `false` . (20)

   Svar: Detta är bara en enkel variant av det extensionella stoppproblemet. Om vi har en funktion `halt` enl ovan skulle vi kunna definiera en funktion `termcnv` genom

   ```
   termcnv f i = if (halt f i) then loop else 1
   ```

   som har egenskapen att `termcnv f i` terminerar om och endast om `f i` ej terminerar. Detta gäller för alla funktioner `f`. Speciellt för den funktion `strange` som är definierad av

   ```
   strange i = termcnv strange i
   ```

   Definitionen av `strange` visar att `termcnv strange i` terminerar samtidigt som `strange i`, vilket motsäger egenskapen ovan.

6. Antag att vi skulle använda programspråket Java (ni kan byta ut Java mot Ada, Pascal, Basic, Fortran eller Cobol) som beräkningsmodell!

   - Vad betyder det att en partiell funktion $\mathtt{f} \in \mathbf{N} \tilde{\to} \mathbf{N}$ är Java-beräkningsbar? (15)

     Svar: Vi måste först ha ett sätt att representera naturliga tal i Java, låt $\ulcorner \mathtt{n} \urcorner$ vara det värde i Java som representerar det naturliga talet $\mathtt{n}$. En funktion $\mathtt{f} \in \mathbf{N} \tilde{\to} \mathbf{N}$ är Java-beräkningsbar om och endast om det finns en funktion (procedur, metod eller något annat som representerar funktioner) $\mathtt{p}$ i Java så att när man applicerar (anropar) $\mathtt{p}$ på $\ulcorner \mathtt{n} \urcorner$ får värdet $\ulcorner \mathtt{f}(\mathtt{n}) \urcorner$ om och endast om $\mathtt{f}(\mathtt{n})$ är definierad.

   - Beskriv vad som behöver göras för att bevisa att den operationella semantiken till Java är Java-beräkningsbar! (15)

     Svar: Först måste man definiera den abstrakta syntaxen för Java, sedan den operationella semantiken, dvs en relation $\mathtt{p} \longrightarrow \mathtt{q}$, vilken är sann om Java-programmet $\mathtt{p}$ beräknar till värdet $\mathtt{q}$. Värdet kan vara tämligen komplicerat då det måste innehålla en beskriving av programmet $\mathtt{p}$'s interaktion med omvärlden. Sen måste vi ge ett sätt att representera ett godtycklig Java-program $\mathtt{p}$ som ett Java-värde $\ulcorner \mathtt{p} \urcorner$. Sen måste vi ge en självinterpretator i Java, dvs ett program **eval** sådant att

     $$(\mathbf{eval}\ \ulcorner \mathtt{p} \urcorner) \longrightarrow \ulcorner \nu \urcorner \ \text{ om och endast om } \ \mathtt{p} \longrightarrow \nu$$

7. Antag att $\ulcorner \mathtt{if} \urcorner$ representerar $\mathtt{if}$-funktionen, $\ulcorner \mathtt{iszero} \urcorner$ representerar $\mathtt{iszero}$-funktionen, $\ulcorner \mathtt{succ} \urcorner$ representerar efterföljarfunktionen, $\ulcorner \mathtt{0} \urcorner$ representerar talet $\mathtt{0}$, $\ulcorner \mathtt{mult} \urcorner$ representerar multiplikation och $\ulcorner \mathtt{pred} \urcorner$ representerar predecessor-funktionen i λ-kalkyl.

   Varför kan vi inte införa följande definition av fakultetsfunktionen i λ-kalkyl:

   $$\begin{aligned} \mathtt{fac} = \lambda \mathtt{n}.&\ulcorner \mathtt{if} \urcorner (\ulcorner \mathtt{iszero} \urcorner\ \mathtt{n}) \\ &(\ulcorner \mathtt{succ} \urcorner\ \ulcorner \mathtt{0} \urcorner) \\ &(\ulcorner \mathtt{mult} \urcorner\ \mathtt{n}\ (\mathtt{fac}(\ulcorner \mathtt{pred} \urcorner\ \mathtt{n}))) \end{aligned}$$

   Visa hur man uttrycker funktionen $\mathtt{fac}$ på ett riktigt sätt i λ-kalkyl! (20)

   Svar: Se läroboken sid 175.

Lycka till!