Göteborgs Universitet och Chalmers Tekniska Högskola — 19 januari 2005
Datavetenskap — INN111/TIN120

Tentamen i Beräkningsmodeller

Lördagen den 1 februari 2003, kl 8.45 – 12.45.
Ansvarig lärare: Bengt Nordström, tel 0708 - 96 69 14.
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 7 februari kl 11.00 i Bengt Nordströms tjänsterum. Lösningar till tentan kommer att finnas tillgängliga från kursens hemsida.

1. Vad säger Church-Rossers sats? (10)

   Svar: Om M och N är normalformen till ett program i λ-kalkyl så är M och N alfa-konvertibla.

2. I λ-kalkyl byggs uttryck upp med hjälp av variabler, abstraktion och applikation. Trots att detta verkar primitivt i överkant kan ju enligt Churchs tes alla beräkningsbara funktioner uttryckas i λ-kalkyl.

   (a) Hur uttrycks talet 3 i λ-kalkyl? (10)

   (b) Ge ett exempel på ett program i λ-kalkyl som inte terminerar. (10)

   (c) Fungerar det exemplet i Haskell (som också har variabler, abstraktioner och applikationer)? (2)

   Svar: Ett vanligt sätt att representera 3 är $\lambda s.\lambda z.s(s(sz))$.
   Ett exempel på ett program som inte terminerar är $(\lambda x.x\ x)\ (\lambda x.x\ x)$.
   Det har ett redex och det reducerar till sig själv. Det programmet är inte typbart i Haskell, eftersom argumentet x både används som argument och funktion, om andra förekomsten av x har typ A så måste första förekomsten ha typ $A \to B$ för något B. Men båda förekomsterna av x måste ju ha samma typ och därför måste $A = A \to B$, vilket är omöjligt.

3. Ange om följande påståenden är sanna eller falska samt ge ett bevis för detta!

   (a) Mängden av alla uttryck i λ-kalkyl som har en normalform är uppräknelig. (15)

   Svar: Sant. Mängden av alla uttryck är uppräknelig, eftersom mängden av alla strängar är uppräknelig. Alla delmängder till en uppräknelig mängd är uppräknelig.

   (b) Alla positiva rationella tal $\mathbf{Q}_+$ (mängden av tal som kan skrivas som $\frac{i}{j}$ där i och j är naturliga tal, $j \neq 0$) är uppräkningsbar. (15)

   Svar: Sant. Den här mängden liknar ju mängden $\mathbf{N} \times \mathbf{N}$ som är uppräknelig. Skillnaden mellan mängderna är att vissa element saknas i $\mathbf{Q}_+$ (nämnaren måste vara positiv och det finns flera bråktal som är lika fastän de är uppbyggd av olika talpar). Vi kan använda samma teknik som i övningshäftet sidan 11, eller också kan vi konstruera en total injektiv funktion $f \in \mathbf{Q}_+ \to \mathbf{N}$ genom

   $$f(\frac{i}{j}) = 2^{i+1} * 3^j \text{där i och j saknar gemensam delare och } j > 0$$

   Denna är total eftersom den avbildar alla bråktal och den är injektiv enligt aritmetikens fundamentalsats. Notera kravet att i och j saknar gemensam delare, annars kommer inte f att vara en funktion (den kommer ju i så fall att avbilda $\frac{1}{2}$ och $\frac{2}{4}$ på olika tal.)

4. En partiell funktion i $\mathbf{N} \tilde{\to} \mathbf{N}$ är Turing-beräkningsbar om det finns en Turing-maskin som beräknar den. Ge en förklaring av detta! Dvs förklara vad det betyder att en Turing maskin beräknar en funktion. (10)

   Svar: Se boken sid 205

5. Ge ett exempel på ett program i språket χ som terminerar till svag huvud normal form, men vars fullständiga evaluering ej terminerar. Motivera! (10)

   Svar: s Ω, där s är en konstruerare och Ω är ett icketerminerande program

6. Det finns fem programkonstruktioner i språket **PRF**, de första fyra har en syntax som kan beskrivas informellt på följande sätt: (40)

$$\begin{aligned}
\mathbf{z} &\in \mathbf{PRF}_0 \\
\mathbf{s} &\in \mathbf{PRF}_1 \\
\mathbf{proj}(n,i) &\in \mathbf{PRF}_{n+1} \text{ if } i \leq n \\
\mathbf{comp}(g,f_1,\ldots,f_m) &\in \mathbf{PRF}_n \text{ if } g \in \mathbf{PRF}_m, f_i \in \mathbf{PRF}_n, 1 \leq i \leq m
\end{aligned}$$

och semantiken beskrivs informellt som:

$$\begin{aligned}
\mathbf{z}() &= 0 \\
\mathbf{s}(j) &= j+1 \\
\mathbf{proj}(n,i)(j_0,\ldots,j_n) &= j_i \\
\mathbf{comp}(g,f_1,\ldots,f_m)(j_1,\ldots,j_n) &= g(f_1(j_1,\ldots,j_n),\ldots,f_m(j_1,\ldots,j_n))
\end{aligned}$$

Ge en informell beskrivning av den konstruktion som saknas!

Visa också hur man kan uttrycka fakultetsfunktionen som definieras av att $f(0) = 1$ och $f(n) = 1*2*\cdots n$ för alla naturliga tal $n > 0$. För den sista uppgiften är det viktigt att du motiverar svaret, dvs antingen visa att programmet verkligen uppfyller de ekvationer som skall gälla för fakultetsfunktionen eller också visa att ditt sätt att komma fram till programmet är sådant att programmet är korrekt. Det räcker alltså inte att bara ge programmet. Du kan anta att multiplikationsfunktionen redan är definierad.

> Svar: Den konstruktion som saknas är operatorn för primitiv rekursion med syntax:
>
> $$\mathbf{rec}(g,h) \in \mathbf{PRF}_{n+1} \text{ if } g \in \mathbf{PRF}_n, h \in \mathbf{PRF}_{n+2}$$
>
> vars semantik beskrivs informellt som:
>
> $$\begin{aligned}
> \mathbf{rec}(g,h)(0,j_1,\ldots,j_n) &= g(j_1,\ldots,j_n) \\
> \mathbf{rec}(g,h)(y+1,j_1,\ldots,j_n) &= h(y,\mathbf{rec}(g,h)(y,j_1,\ldots,j_n),j_1,\ldots,j_n)
> \end{aligned}$$
>
> Vi ska nu skriva ett program f för fakultetsfunktionen. Om vi försöker uttrycka f på en primitivt rekursiv form ser vi att
>
> $$\begin{aligned}
> f(0) &= 1 \\
> f(n+1) &= (n+1) * f(n)
> \end{aligned}$$
>
> Om vi ansätter
>
> $$f =_{\text{def}} \mathbf{rec}(g,h)$$

där $g \in \mathbf{PRF}_0$ och $h \in \mathbf{PRF}_2$, så vet vi att $g[0] = 1$ måste gälla. Det är uppfyllt om

$$g =_{\text{def}} \mathbf{comp}(\mathbf{s}, [\mathbf{z}]).$$

Vi vet att följande skall gälla för funktionen h:

$$\begin{aligned} f[n+1] &= \mathbf{rec}(g, h)[n+1] \\ &= h[n, f(n)] \\ &= (n+1) * f(n) \end{aligned}$$

Vi vill alltså konstruera ett program h i $\mathbf{PRF}_2$ så att $h[n, f(n)] = (n+1) * f(n)$. Detta är uppfyllt om funktionen h uppfyller $h[n, m] = \mathbf{mul}[n+1, m]$, där **mul** är det program som utför multiplikation.
Om vi nu försöker ansätta att h måste ha formen

$$h = \mathbf{comp}(\mathbf{mul}, [e_1, e_2])$$

för några program e_1 och e_2, så vet vi att följande måste gälla:

$$\begin{aligned} \mathbf{comp}(\mathbf{mul}, [e_1, e_2])[n, m] &= \mathbf{mul}(e_1[n, m], e_2[n, m]) \\ &= \mathbf{mul}(n+1, m). \end{aligned}$$

Detta är uppfyllt om

$$\begin{aligned} e_1[n, m] &= n+1 \\ e_2[n, m] &= m. \end{aligned}$$

Detta är uppfyllt om e_2 är en projektion, nämligen

$$e_2 =_{\text{def}} \mathbf{proj}(,1)^1$$

och om e_1 är en komposition:

$$e_1 =_{\text{def}} \mathbf{comp}(\mathbf{s}, \mathbf{proj}(,1)^0)$$

ty $\mathbf{comp}(\mathbf{s}, \mathbf{proj}(,1)^0)[n, m] = \mathbf{s}(\mathbf{proj}(,1)^0[n, m]) = n+1$
För att sammanfatta så kan vi alltså definiera fakultetsfunktionen som

$$\begin{aligned} f =_{\text{def}} \mathbf{rec}(&\mathbf{comp}(\mathbf{s}, \mathbf{z}), \\ &\mathbf{comp}(\mathbf{mul}, [\mathbf{comp}(\mathbf{s}, \mathbf{proj}(,1)^0), \\ &\quad \mathbf{proj}(,1)^1])) \end{aligned}$$

7. Bevisa att man i Haskell (eller något annat 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 stopp-problemet. 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.

Lycka till!