Göteborgs Universitet och Chalmers Tekniska Högskola — 19 januari 2005
Datavetenskap — TDA180/TDA181/INN110

Tentamen i kurserna Beräkningsmodeller (TDA181/INN110) och Grundläggande Datalogi (TDA180)

Onsdagen den 12 Januari 2005, kl 14.00 – 18.00 i V-huset.
Ansvarig lärare: Bengt Nordström, tel 0730-79 42 89 (fram till kl 16.45).
Tillåtna hjälpmedel: Inga.
Börja varje uppgift på nytt blad. Skriv endast på en sida av papperet. Varje svar skall motiveras! Komplicerade lösningar och motiveringar kan ge poängavdrag.

Kursen är värd 4 p vid Chalmers och 5 p vid universitetet. Detta förklarar följande betygsgränser: CTH: 3=80p, 4=100p, 5=120p, GU: G=100p, VG=150p. Elever vid universitetet kan tillgodogöra sig poäng från hemuppgifter inlämnade under 2004.

Examensvisning kommer att äga rum måndagen den 24 januari kl 11.00 i Bengt Nordströms tjänsterum. Lösningar till tentan kommer att finnas tillgängliga från kursen Beräkningsmodellers hemsida.

1. Vad säger Churchs tes? Varför heter den inte Churchs sats? Ge några skäl för att tro att den är sann! (10)

   Svar:

2. (a) Ge ett exempel på ett program i λ-kalkyl som inte terminerar. (10)
   (b) Fungerar det exemplet i Haskell? (5)

   Svar: 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. (a) Vad är en fixpunktskombinator? (5)

   (b) Ge ett exempel på en fixpunktskombinator och visa att det är en sådan! (10)

   (c) Varför är fixpunktskombinatorer viktiga? (5)

5. 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

6. Det finns fem programkonstruktioner i språket **PRF**, de primitivt rekursiva funktionerna. 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. Lös en av följande uppgifter:

   (a) Följande uppgift är för de som har studerat språket χ:

      i. 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

      ii. 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` . (15)

         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.

   (b) Följande uppgift är för de som studerat PCF:

      i. Define a PCF-program that behaves like the following Haskell program: (9)

```
ack 0 m = S m
ack (S n) 0 = ack n (S 0)
ack (S n) (S m) = ack n (ack (S n) m)
```

         Svar:

```
ack = fix (\f. \n. \m. if (isZero n)
                          (Succ m)
                          (if (isZero m)
                              (f (pred n) (Succ Zero))
                              (f (pred n) (f n (pred m)))))
```

ii. Give the small steps semantics rules for fix and if. (6)
Remember that there are three rules involving if!
Svar:

```
Let -> represent 1 step reduction

 For Fix Points Combinator:

   fix f -> f (fix f)

 For Conditionals:

          b -> b'
  ------------------------
   If b d e -> If b' d e

   If True d e -> d

   If False d e -> e
```

iii. Apply your program to the values Succ Zero and Succ Zero and show, using the small steps semantics of PCF, how to reduce the program to its value. (10)
In order to simplify your answer, you can use the normal decimal representation of the Natural numbers, that is, you can simply write 2 instead of Succ(Succ Zero). In addition, you do not have to write all the steps in the reduction sequence but enough steps so that one can still follow the derivation.
Svar:

```
ack 1 1 ->* if (isZero 1)
               2
               (if (isZero 1) (ack 0 1) (ack 0 (ack 1 0)))
->* if (isZero 1) (ack 0 1) (ack 0 (ack 1 0))
->* ack 0 (ack 1 0)
->* if (isZero 0)
       (Succ (ack 1 0))
       (if (isZero (ack 1 0))
```

```
            (ack 0 1)
            (ack 0 (ack 0 (pred (ack 1 0)))))
->* Succ (ack 1 0)
->* Succ (if (isZero 1)
               1
               (if (isZero 0) (ack 0 1) (ack 0 (ack 1 0))))
->* Succ (if (isZero 0) (ack 0 1) (ack 0 (ack 1 0)))
->* Succ (ack 0 1)
->* Succ (if (isZero 0)
               2
               (if (isZero 1) (ack 0 1) (ack 0 (ack 0 0))))
->* Succ 2 -> 3
```

Lycka till!