Göteborgs Universitet och Chalmers Tekniska Högskola — 27 maj 2007
Datavetenskap — TDA180/TDA181/INN110

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

1 Juni 2007, kl 8.30 ++ 12.30 i 14.00 – 17.00 i M-huset.
Ansvarig lärare: Bengt Nordström, tel 0730-79 42 89.
Tillåtna hjälpmedel: Inga.
Börja varje uppgift på nytt blad. Skriv endast på en sida av papperet. Varje svar skall motiveras! Svar utan motivering ger inga poäng. Komplicerade lösningar och motiveringar kan ge poängavdrag.

Poäng från hemuppgifter inlämnade under 2006 kan tillgodoräknas. Betygsgränser för 5 p: CTH: 3=80p, 4=120p, 5=160p, GU: G=80p, VG=140p.

Examensvisning kommer att äga rum fredagen den 8 juni 13.30 – 13.45 i Bengt Nordströms tjänsterum. Lösningar till tentan kommer att finnas tillgängliga från kursen Beräkningsmodellers hemsida.

1. Funktionen `iter` definierad av

   ```
   iter n f a = f (f ... (f a)...)
   ```

   där det finns n st förekomster av f i höger led kan definieras med hjälp av operatorn `rec` för primitiv rekursion. Visa detta! Operatorn `rec` definieras av följande

   ```
   rec 0     d e = d
   rec (n+1) d e = e n (rec n d e)
   ```

2. Låt f vara en beräkningsbar funktion från de naturliga talen till de naturliga talen. Bevisa att det finns oändligt många syntaktiskt olika λ-termer som beräknar f! (Två λ termer är syntaktiskt lika om de är alfa-konvertibla.).

   Svar: Identitetsfunktionen är beräkningsbar, t.ex. i λ-kalkyl kan den beräknas av $\mathsf{I} =_{\mathsf{def}} \lambda x.x$. Om f beräknas av programmet F så beräknas f också av $\lambda x.\mathsf{I}(\ldots \mathsf{I}(\mathsf{F}\ x)\ldots)$, för godtyckligt många förekomster av I.

3. Trots resultatet i föregående uppgift finns det strikt fler matematiska funktioner i $\mathbf{N} \to \mathbf{N}$ än det finns beräkningsbara funktioner i $\mathbf{N} \to \mathbf{N}$. Bevisa detta genom att lösa följande uppgifter:

   (a) Definiera vad det betyder att en funktion i $\mathbf{N} \to \mathbf{N}$ är beräkningsbar i λ-kalkyl.

   Svar: En funktion f är beräkningsbar om det finns en λ term F så att

   $$\mathsf{F}\ulcorner \mathfrak{n} \urcorner \to \ulcorner \mathfrak{m} \urcorner$$

   om och endast om

   $$\mathsf{f}(\mathfrak{n}) = \mathfrak{m}$$

   , där $\ulcorner \mathfrak{n} \urcorner$ är koden för $\mathfrak{n}$ i λ-kalkyl.

   (b) Bevisa att mängden av program är uppräkneligt (oberoende av programmeringsspråk).

   (c) Bevisa att mängden $\mathbf{N} \to \mathbf{N}$ ej är uppräknelig.

   Svar: Se föreläsningsanteckningarna "Are all functions computable?"

4. Bevisa eller motbevisa följande påståenden:

   (a) Funktionen som är odefinierad för varje argument är beräkningsbar. (5)

   Svar: Funktionen är beräkningsbar eftersom den kan beräknas av ett icke terminerande program, t.ex. $(\lambda \mathsf{x}.(\mathsf{x}\,\mathsf{x})(\mathsf{x}\,\mathsf{x}))(\lambda \mathsf{x}.(\mathsf{x}\,\mathsf{x})(\mathsf{x}\,\mathsf{x}))$ i lambda-kalkyl.

   (b) Om M är normalformen av N (i lambda-kalkyl) och N är öppet, så är M öppet. (5)

   Svar: Detta gäller inte, låt N vara det öppna uttrycket $(\lambda\ \mathsf{x}.(\lambda\ \mathsf{y}.\mathsf{y}))\mathsf{z}$, som har variabeln z fri. Då blir M uttrycket $\lambda\ \mathsf{y}.\mathsf{y}$ som är slutet. (Vi låter allstå N vara en funktion som inte beror på sitt argument, och applicerar det på ett öppet uttryck.)

   (c) Alla slutna uttryck i lambda-kalkyl har en normalform och denna normalform är unik. (5)

   Svar: Nej, det finns ju uttryck i lambda-kalkyl som inte terminerar, t.ex. $\mathsf{T}\,\mathsf{T}$, där T är uttrycket $\lambda\mathsf{x}.(\mathsf{x}\,\mathsf{x})$

   (d) Mängden av totala funktioner från $\mathbf{N}$ till **Bool** är uppräknelig. (15)

Svar: Falskt. Mängden är inte uppräkningsbar, om de vore det skulle det finnas en uppräkning $\mathtt{f}_\mathtt{i}$ av dem. Men uppräkningen kan inte innehålla funktionen $\mathtt{d}$ som är definierad av

$$\mathtt{d}(\mathtt{n}) = \neg \mathtt{f}_\mathtt{n}(\mathtt{n})$$

eftersom den skiljer sig från alla funktioner i uppräkningen. Om den skulle vara lika med funktionen $\mathtt{f}_\mathtt{j}$ skulle $\mathtt{d}(\mathtt{i}) = \mathtt{f}_\mathtt{j}(\mathtt{i})$ för alla $\mathtt{i}$. Men detta gäller inte för $\mathtt{i} = \mathtt{j}$.

(e) Mängden av totala funktioner från **Bool** till **N** är uppräknelig. (15)

Svar: Ja. Intuitionen är att mängen har samma kardinalitet som $\mathbf{N} \times \mathbf{N}$, mängden av par av naturliga tal. Ett element $\mathtt{f}$ i funktionsmängden är ju fullständigt beskriven av två tal, funktionens värde för de två boolska elementen. För att bevisa att mängden är uppräkningsbar räcker det att ge en total och injektiv funktion $\mathtt{g} \in (\mathbf{Bool} \to \mathbf{N}) \to \mathbf{N}$. En sådan är:

$$\mathtt{g}(\mathtt{f}) = 2^{\mathtt{f}(\mathbf{true})+1} * 3^{\mathtt{f}(\mathbf{false})+1}$$

Den är ju total (definitionen gäller för alla $\mathtt{f}$) och injektiv. Om $\mathtt{g}(\mathtt{f}) = \mathtt{g}(\mathtt{f}')$, så gäller

$$2^{\mathtt{f}(\mathbf{true})+1} * 3^{\mathtt{f}(\mathbf{false})+1} = 2^{\mathtt{f}'(\mathbf{true})+1} * 3^{\mathtt{f}'(\mathbf{false})+1}$$

Men eftersom primtalsuppdelningen är unik måste $\mathtt{f}(\mathbf{true}) = \mathtt{f}'(\mathbf{true})$ och $\mathtt{f}(\mathbf{false}) = \mathtt{f}'(\mathbf{false})$, dvs $\mathtt{f} = \mathtt{f}'$.

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

   (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:

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

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.

6. (a) Vad är en fixpunktskombinator? (5)

   (b) Ge ett exempel på en fixpunktskombinator och visa att det är en sådan! (5)

   (c) Varför är fixpunktskombinatorer viktiga? (5)

   Svar: Se läroboken!

7. 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

$$e_2 =_{\mathsf{def}} \mathbf{proj}(2, 1)^1$$

och om e_1 är en komposition:

$$e_1 =_{\mathsf{def}} \mathbf{comp}(\mathbf{s}, \mathbf{proj}(2, 1)^0)$$

ty $\mathbf{comp}(\mathbf{s}, \mathbf{proj}(2, 1)^0)[n, m] = \mathbf{s}(\mathbf{proj}(2, 1)^0[n, m]) = n + 1$
För att sammanfatta så kan vi alltså definiera fakultetsfunktionen som

$$\begin{aligned}f =_{\mathsf{def}} \mathbf{rec}(&\mathbf{comp}(\mathbf{s}, \mathbf{z}), \\ &\mathbf{comp}(\mathbf{mul}, [\mathbf{comp}(\mathbf{s}, \mathbf{proj}(2, 1)^0), \\ &\quad \mathbf{proj}(2, 1)^1]))\end{aligned}$$

8. Lös en av följande uppgifter (beroende på om du studerat χ eller PCF):

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

      i. Skriv ett program prod i χ (utan syntaktiskt socker) som är definierat så att (8)

$$\mathsf{prod}\ n = (0 + 1) * (1 + 2) * \ldots * (n - 1 + n)$$

      Du kan anta att vi har definierat funktionerna add och mult som utför additon respektive multiplikation. Förklara hur du representerar de naturliga talen (om du inte vill behöver du inte använda standard-representationen).

ii. Bevisa (med induktion) att ovanstående gäller! (12)

Svar: Vi använder standard-representationen av tal, dvs talet 0 representeras av $\mathsf{zero}\langle\rangle$ och talet $\mathfrak{n}+1$ av $\mathsf{succ}\langle\mathfrak{n}'\rangle$, där $\mathfrak{n}'$ är representationen av $\mathfrak{n}$. Vi ser att programmet skall uppfylla följande ekvationer:

$$\begin{aligned}\mathsf{prod}\ (\mathsf{zero}\langle\rangle) &= \mathsf{one}\\ \mathsf{prod}\ (\mathsf{succ}\langle\mathfrak{n}\rangle) &= (\mathsf{mult}\ (\mathsf{prod}\ \mathfrak{n}))(\mathsf{add}\ \mathfrak{n}\ (\mathsf{succ}\ \mathfrak{n}))\end{aligned}$$

Låt oss införa förkortingen

$$\mathsf{f}\ \mathfrak{m}\ \mathfrak{n} = (\mathsf{mult}\ \mathfrak{m})(\mathsf{add}\ \mathfrak{n}\ (\mathsf{succ}\ \mathfrak{n}))$$

Vi skall alltså lösa ekvationerna

$$\begin{aligned}\mathsf{prod}\ (\mathsf{zero}\langle\rangle) &= \mathsf{one}\\ \mathsf{prod}\ (\mathsf{succ}\langle\mathfrak{n}\rangle) &= \mathsf{f}\ (\mathsf{prod}\ \mathfrak{n})\ \mathfrak{n}\end{aligned}$$

Vi kan förenkla problemet till att lösa ekvationen

$$\begin{aligned}\mathsf{prod}\ z = \ &\mathbf{case}\ z\ \mathbf{of}\{\\ &\mathsf{zero}\langle\rangle : \mathsf{one},\\ &\mathsf{succ}\langle\mathfrak{n}\rangle : \mathsf{f}\ (\mathsf{prod}\ \mathfrak{n})\ \mathfrak{n}\}\end{aligned}$$

vilken löses av

$$\begin{aligned}\mathsf{prod}\ =_{\mathrm{def}}\ &\mathbf{rec}\ \mathrm{p} = \lambda z \to \mathbf{case}\ z\ \mathbf{of}\{\\ &\mathsf{zero}\langle\rangle : \mathsf{one},\\ &\mathsf{succ}\langle\mathfrak{n}\rangle : \mathsf{f}\ (\mathrm{p}\ \mathfrak{n})\ \mathfrak{n}\}\end{aligned}$$

vilket utan syntaktiskt socker skrives:

$$\begin{aligned}\mathsf{prod}\ =_{\mathrm{def}}\ &\mathbf{rec}\ \mathrm{p} = \lambda z \to \mathbf{case}\ z\ \mathbf{of}\{\\ &\mathsf{zero} : \lambda\ \mathfrak{u} \to \mathsf{one},\\ &\mathsf{succ} : \lambda\ \mathfrak{n} \to \mathsf{f}\ (\mathrm{p}\ \mathfrak{n}.\mathsf{arg1})\ \mathfrak{n}.\mathsf{arg1}\}\end{aligned}$$

Denna definition av konstanten prod är korrekt ty, i basfallet får vi:

```
prod (zero  <>)
= {enl definitionen av prod}
(rec p = \z -> case z of {
```

```
            zero: \u -> one;
            succ: \n -> f (p n.arg1) n.arg1})(zero <>)
= {enl beräkningsregel för rec}
\z -> case z of {
            zero: \u -> one;
            succ: ...}    (zero <>)
= {enl beräkningsregel för applikation}
case zero <> of {
            zero: \u -> one;
            succ: ... })
= {enl beräkningsregel för case}
 (\u -> one) <>
=
one
```

I efterföljarfallet får vi följande beräkningar:

```
prod (succ  <n>)
= {enl definitionen av prod}
(rec p = \z -> case z of {
            zero: \u -> one;
            succ: \n -> f (p n.arg1) n.arg1})(succ <n>)
= {enl beräkningsregel för rec}
\z -> case z of {
            zero: \u -> one;
            succ: \n -> f (prod  n.arg1) n.arg1})(succ <n>)
= {enl beräkningsregel för applikation}
case succ <n> of {
            zero: \u -> one;
            succ: \n -> f (prod  n.arg1) n.arg1})
= {enl beräkningsregel för case}
 (\n -> f (prod  n.arg1) n.arg1)  <n>
=
 f (prod  n) n)
```

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

i. Define a PCF-program that behaves as prod, for any $n > 0$ (8)

$$\mathsf{prod}\ n = (0+1) * (1+2) * \ldots * (n-1+n)$$

ii. What is the output of your PCF-program when applied to

the value Zero? Justify by showing the main steps in the reduction of prod Zero. Explain. Here you should assume that both add and mult have the expected semantics. You can use either big or small semantics. (7)

iii. What is the purpose of the fix operator in PCF? Give a PCF-program that contains no fix operator and that does not terminate. Justify! (5)

Lycka till!