Göteborgs Universitet och Chalmers Tekniska Högskola
Datavetenskap

25 oktober 2005
TDA180/TDA181/INN110

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

Onsdagen den 19 oktober 2005, kl 8.30 – 12.30 i V-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! Komplicerade lösningar och motiveringar kan ge poängavdrag.

Poäng från hemuppgifter inlämnade under 2005 kan tillgodoräknas.

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. Det finns ett fåtal elever på Chalmers som läste motsvarande 3p-kurs. För dem är poänggränserna 3=60p, 4=75p, 5=90p.

Examensvisning kommer att äga rum onsdagen den 2 november kl 11.00 i Bengt Nordströms tjänsterum. Lösningar till tentan kommer att finnas tillgängliga från kursen Beräkningsmodellers hemsida.

1. Bevisa eller motbevisa följande påståenden:

   (a) Funktionen (10)

   $$f(x) = \begin{cases} x & \text{om } x \text{ är udda,} \\ \text{odefinierat} & \text{för övrigt} \end{cases}$$

   är beräkningsbar.

   Svar: Funktionen är beräkningsbar eftersom den kan beräknas av programmet `F x = if (even x) then loop else x`, där programmet `loop` aldrig terminerar och programmet `even` avgör om argumentet är jämnt. Programmet `even` är t.o.m. primitivt rekursivt eftersom det kan skrivas på primitivt rekursiv form: `even 0 = true` och `even (s x) = not (even x)`)

   (b) Om M är normalformen av N (i lambda-kalkyl) och N är öppet, så är M öppet. (15)

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

(c) Om M är ett slutet uttryck så har M en normalform (i lambda-kalkyl).

Svar: Nej, det finns ju uttryck i lambda-kalkyl som inte terminerar, t.ex. $T\ T$, där T är uttrycket $\lambda x.(x\ x)$ (15)

(d) Mängden av totala funktioner från **N** till **Bool** är uppräknelig. (20)

Svar: Falskt. Mängden är inte uppräkningsbar, om de vore det skulle det finnas en uppräkning f_i av dem. Men uppräkningen kan inte innehålla funktionen d som är definierad av

$$d(n) = \neg f_n(n)$$

eftersom den skiljer sig från alla funktioner i uppräkningen. Om den skulle vara lika med funktionen f_j skulle $d(i) = f_j(i)$ för alla i. Men detta gäller inte för $i = j$.

(e) Mängden av totala funktioner från **Bool** till **N** är uppräknelig. (20)

Svar: Ja. Intuitionen är att mängen har samma kardinalitet som $\mathbf{N} \times \mathbf{N}$, mängden av par av naturliga tal. Ett element 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 $g \in (\mathbf{Bool} \to \mathbf{N}) \to \mathbf{N}$. En sådan är:

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

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

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

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

2. 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? (14)

(b) Är det väsentligt att funktionen skall vara surjektiv? (14)

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:

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

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.

3. (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)

Svar: Se läroboken!

4. 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 (10)

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

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 $\mathtt{n}+1$ av $\mathsf{succ}\langle\mathtt{n}'\rangle$, där $\mathtt{n}'$ är representationen av $\mathtt{n}$. Vi ser att programmet skall uppfylla följande ekvationer:

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

Låt oss införa förkortingen

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

Vi skall alltså lösa ekvationerna

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

Vi kan förenkla problemet till att lösa ekvationen

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

vilken löses av

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

vilket utan syntaktiskt socker skrives:

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

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 $n > 0$ (10)

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

You can assume that you already have two PCF-programs add and mult performing addition and multiplication of Natural numbers, respectively.

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!