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

Tentamen i Beräkningsbarhet och lambda-kalkyl

Måndagen den 10 januari 1999, kl 8.45 – 13.45 i sal VV.
Ansvarig lärare: Bengt Nordström, tel 1033, eller 55 45 25 (hem)
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 inlämingsuppgifter som har delats ut under kursens gång. För 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 den 28 februari kl 14.15 i MD8.

1. Vad menas med att en funktion är Turing-beräkningsbar? (5)

   Svar: sid 205 i boken

2. Hur formulerar man och bevisar stopp-problemet för Turing-maskiner? (15)

   Svar: Anta att vi kan lösa stopp-problemet, dvs. att den finns en Turing-maskin X med egenskapen

   $$< \mathsf{X}, (\overline{\mathsf{T}}, \alpha) > \Rightarrow \begin{cases} 1 & \text{om } (\mathsf{T}, \alpha) \text{ terminerar,} \\ 0 & \text{för övrigt} \end{cases}$$

   Vi låter $(\overline{\mathsf{T}}, \alpha)$ vara en positionerad remsa som kodar paret av $\overline{\mathsf{T}}$ och α.

   Definiera nu en ny Turing-maskin Y genom att lägga till instruktioner till X på följande sätt: Om $< \mathsf{X}, (\overline{\mathsf{T}}, \alpha) >$ stannar på remsan 1 så låter vi Y fortsätta att gå åt höger i all oändlighet. Annars låter vi Y terminera.

   Vi får alltså att $< \mathsf{Y}, (\overline{\mathsf{T}}, \alpha) >$ terminerar om och endast om $< \mathsf{T}, \alpha >$ ej terminerar.

   Nu konstruerar vi en ny Turing-maskin genom att lägga till instruktioner till Y på följande sätt. Låt Z först kopiera sitt indata-band β till (β, β) och exekvera sedan Y.

   Terminerar $< \mathsf{Z}, \overline{\mathsf{Z}} >$? Den kommer att uppföra sig som maskinen $< \mathsf{Y}, (\overline{\mathsf{Z}}, \overline{\mathsf{Z}}) >$. Men $< \mathsf{Y}, (\overline{\mathsf{Z}}, \overline{\mathsf{Z}}) >$ terminerar ej om $< \mathsf{Z}, \overline{\mathsf{Z}} >$ terminerar. Och $< \mathsf{Y}, (\overline{\mathsf{Z}}, \overline{\mathsf{Z}}) >$ terminerar om $< \mathsf{Z}, \overline{\mathsf{Z}} >$ ej terminerar ! Motsägelse. Vårt antagande att vi kan lösa stopp-problemet måste alltså vara felaktigt.

3. Kan man räkna upp alla totala funktioner från $\mathbf{N}$ till $\{1\}$? Motivera! (10)

Svar: Det finns bara en funktion, nämligen den funktion som alltid ger resultatet 1. Mängden av funktioner är alltså ändlig och således uppräknelig.

4. Kan man räkna upp alla partiella funktioner från $\mathbf{N}$ till $\{1\}$? Motivera!

   Svar: Det kan man inte. Om man kunde det fanns det en total surjektiv funktion $F \in \mathbf{N} \to \mathbf{N} \tilde{\to} \{1\}$ som räknar upp dessa. Men funktionen g definierad av

   $$g(n) = \begin{cases} 1 & \text{om } F(n)(n) \text{ är odefinierad,} \\ \text{odefinerad} & \text{för övrigt} \end{cases}$$

   är inte med i uppräkningen. Om den vore det skulle g vara lika med funktionen $F(i)$ för något i. Men detta leder till en motsägelse eftersom vi vet att $g(i) \neq F(i)(i)$. (10)

5. Låt A vara mängden $\{\mathbf{sur}, \mathbf{tot}, \mathbf{inj}\}$ och låt B vara mängden av partiella funktioner från $\mathbf{N}$ till $\mathbf{N}$. Definiera nu en relation P mellan B och A på följande sätt: Funktionen f är P-relaterad till elementet **sur** om f är surjektiv, den är P-relaterad till **tot** om den är total och P-relaterad till **inj** om den är injektiv. Låt nu A' beteckna mängden av alla delmängder av A. Vi definierar sen en relation $P3$ mellan B och A' på följande sätt: Funktionen g är $P3$-relaterad till elementet U om det för alla element x i U gäller att f är P-relaterad till x.

   (a) Relationen $P3$ är inte en partiell funktion från B till A, varför?

   Svar: En funktion som är (t.ex.) både surjektiv och total är ju relaterad till både $\{\mathbf{sur}\}$, $\{\mathbf{tot}\}$

   och $\{\mathbf{sur}, \mathbf{tot}\}$.

   (b) Definiera en ny relation $P'3$ mellan B och A' som blir en funktionell version av $P3$.

   Svar: Vi måste se till att varje funktion blir relaterad till högst en delmängd. Ett naturligt sätt att definiera den vore att relatera varje funktion f enbart till den största delmängden U för vilken f är $P3$-relaterad. Ett annat sätt att uttrycka det är att f är $P3$-relaterad till mängden U om det för alla element x i U gäller att f är P-relaterad till x samt att f inte är relaterad till något element utanför U. (20)

6. (a) Ge en induktiv definition av abstrakta syntaxen för programmen i λ-kalkyl.

   Svar: Vi har tre sätt att bilda uttryck i lambda-kalkyl. Applikation, abstraktion och variabler. Den konkreta syntaxen kan ju beskrivas som $E := (EE)|x|\lambda x.E$. Vi får då följande induktiva definition av den abstrakta syntaxen, vi definierar

mängden Λ, som beror på en mängd **Id** som representerar variabler (Jämför definitionen av χ s abstrakta syntax!):

$$\begin{aligned} \mathbf{apply}(e_1,\ e_2) \in \Lambda \quad &\text{om} \quad e_1, e_2 \in \Lambda \\ \mathbf{lambda}(i, e) \in \Lambda \quad &\text{om} \quad i \in \mathbf{Id}, e \in \Lambda \\ \mathbf{var}(i) \in \Lambda \quad &\text{om} \quad i \in \mathbf{Id} \end{aligned}$$

(b) Blir detta en uppräknelig mängd?

Svar: Om mängden av identifierare är uppräknelig så blir mängden uppräknelig. Mängden Λ är ju bara en blandning av binära träd (applikationer) och listor (abstraktionerna). Anta att g' är en injektion från **Id** till **N**. Vi vill konstruera g, en injektion från Λ till **N**:

$$\begin{aligned} g(\mathbf{apply}(e_1,\ e_2)) &= 2^2 \cdot 3^{g(e_1)} \cdot 5^{g(e_2)} \\ g(\mathbf{lambda}(i, e)) &= 2^3 \cdot 3^{g'(i)} \cdot 5^{g(e)} \\ g(\mathbf{var}(i)) &= 2^4 \cdot 3^{g'(i)} \end{aligned}$$

Detta blir en injektion, om $g(a) = g(b)$ så måste $g(a)$ och $g(b)$ ha samma 2-exponent. Om denna är 2 vet vi tre saker. Dels att a och b har samma yttre form (nämligen **apply**) samt att 3 - och 5-exponenterna är identiska. Men 3 - och 5-exponenterna är koden av argumenten till **apply**. Om 2-exponenten är 3 vet vi att den yttre formen är **lambda**, och att 3 - och 5-exponenten ger delarna. Slutligen, om 2-exponenten är 4 vet vi att att den yttre formen är **var**, etc.

(c) Definiera en förenklad substitutionsoperation för λ-kalkyl. Värdet av funktionen $\mathtt{closedsubst}(e, x, e')$ skall vara resultatet av att substituera uttrycket e' för all fria förekomster av variabeln x i uttrycket e. Uttrycket e' skall vara slutet.

Svar: Detta blir samma definition som definition av substitution i språket χ, se föreläsningsanteckningarnas appendix. Ta bara med klausuler för applikation, variabel och abstraktion.

(d) Är $\mathtt{closedsubst}(e, x, e')$ alltid öppet om e är öppet?

Svar: Nej, variablerna i e kan ju bindas vid substitutionen, låt t.ex. uttrycket e stå för uttrycket x. Då kommer ju alla fria förekomster av variabeln x att bytas ut mot det slutna uttrycket e', dvs resultatet av substitutionen kommer att bli e'. (30)

7. Visa hur man skall definiera språket PRL(k), mängden av primitivt rekursiva funktioner över heltalslistor. Ett element i mängden skall stå för en funktion som tar k stycken heltalslistor som argument och returnerar

en heltalslista som resultat. Du skall ge den abstrakta syntaxen och den informella semantiken. (30)

Kommentar: Språket skall definieras analogt med språket $\mathbf{PRF}_n$, mängden av primitivt rekursiva funktioner med n argument. Varje argument är ett naturligt tal. Det vill säga ett element f i mängden $\mathbf{PRF}_n$ står för en funktion i $\underbrace{\mathbf{N} \times \ldots \times \mathbf{N}}_{n \text{ stycken}} \to \mathbf{N}$. Elementen i mängden $\mathbf{PRF}_n$ byggs upp induktivt enligt följande klausuler:

$$\begin{aligned}
\mathbf{z} &\in \mathbf{PRF}_1 \\
\mathbf{s} &\in \mathbf{PRF}_1 \\
\mathbf{proj}_i^n &\in \mathbf{PRF}_n \quad \text{om} \quad 1 \leq i \leq n \\
\mathbf{compose}(g, f_1, \ldots, f_m) &\in \mathbf{PRF}_n \quad \text{om} \quad g \in \mathbf{PRF}_m, f_i \in \mathbf{PRF}_n, 1 \leq i \leq m \\
\mathbf{rec}(g, h) &\in \mathbf{PRF}_{n+1} \quad \text{om} \quad g \in \mathbf{PRF}_n, h \in \mathbf{PRF}_{n+2}
\end{aligned}$$

Svar: $\mathbf{PRF}_n$ är ett språk som uttrycker primitivt rekursiva funktioner över naturliga tal. Om man skall göra en analog definition av PRL(k) så bör man justera definitionen av $\mathbf{PRF}_n$ på tre punkter. De första två klausulerna i den abstrakta syntaxen uttrycker hur naturliga tal är induktivt uppbyggda (med hjälp av konstruerarna **z** och **s**). I PRL(k) motsvaras det av klausuler för uppbyggnad av listor:

$$\begin{aligned}
\mathbf{nil} &\in \mathsf{PRL}(k) \\
\mathbf{cons}(a) &\in \mathbf{PRF}(k) \quad \text{om} a \in \mathbf{N}
\end{aligned}$$

med den informella sematiken:

$$\begin{aligned}
\mathbf{nil}(t) &= [] \\
\mathbf{cons}(a)(t) &= [a; t]
\end{aligned}$$

Klausulerna för projektion och komposition blir helt analoga.
Den sista klausulen i $\mathbf{PRF}_n$ uttrycker primitiv rekursion över naturliga tal. Vi måste ändra denna så att den uttrycker primitiv rekursion över heltalslistor.
Abstrakt syntax:

$$\mathbf{listrec}(g, h) \in \mathsf{PRL}(k+1) \quad \text{om} \quad g \in \mathsf{PRL}(k), h \in \mathbf{PRF}_{n+3}$$

Informell semantik:

$$\begin{aligned}
\mathbf{listrec}(g, h)([], l_1, \ldots, l_n) &= g(l_1, \ldots, l_n) \\
\mathbf{listrec}(g, h)([a, as], l_1, \ldots, l_n) &= h([a], as, l_1, \ldots, l_n, y, \mathbf{listrec}(g, h)(as, l_1, \ldots, l_n, y)
\end{aligned}$$

8. Skriv ett program $\mathbf{isnat}_1$ i språket χ med följande egenskap:

$$(\mathbf{isnat}_1\ n) = \mathbf{true}() \quad \text{om } n \text{ är ett naturligt tal}$$

Försök sedan att skriva ett program $\mathbf{isnat}_2$ med egenskapen:

$$(\mathbf{isnat}_2\ n) = \begin{cases} \mathbf{true}() & \text{om } n \text{ är ett naturligt tal,} \\ \mathbf{false}() & \text{för övrigt} \end{cases}$$

Föreslå en utvidgning av språket så att man lätt kan uttrycka sådana program. Föreslå ny konkret syntax, ny abstrakt syntax och ge (informell och formell) semantik för utvidgningen! (30)

Kommentar: Språket χ har följande konkret syntax:

$(e_1\ e_2)$	applikation
$\lambda x.e$	abstraktion
$c\ e$	konstruerar-applikation
case e **of** $\{c_1 : e_1,\ \ldots\}$	case-uttryck
$[l_1 = e_1;\ \ldots]$	struktur
$e.l$	projektion
rec $x = e$ **end**	rekursion
x	variabel

Den abstrakta syntaxen är följande:

$$\begin{aligned}
\mathbf{apply}(e_1,\ e_2) \in \chi &\quad \text{om} \quad e_1, e_2 \in \chi \\
\mathbf{lambda}(i, e) \in \chi &\quad \text{om} \quad i \in \mathbf{Id}, e \in \chi \\
\mathbf{constr}(i, e) \in \chi &\quad \text{om} \quad i \in \mathbf{Id}, e \in \chi \\
\mathbf{case}(e, t) \in \chi &\quad \text{om} \quad e \in \chi, t \in \mathbf{T}(\mathbf{Id}, \chi) \\
\mathbf{struct}(t) \in \chi &\quad \text{om} \quad t \in \mathbf{T}(\mathbf{Id}, \chi) \\
\mathbf{proj}(e, i) \in \chi &\quad \text{om} \quad e \in \chi, i \in \mathbf{Id} \\
\mathbf{rec}(i, e) \in \chi &\quad \text{om} \quad e \in \chi, i \in \mathbf{Id} \\
\mathbf{var}(i) \in \chi &\quad \text{om} \quad i \in \mathbf{Id}
\end{aligned}$$

Semantiken defineras av:

$$\frac{e_1 \longrightarrow \mathbf{lambda}(i, e_3) \qquad e_3[i \leftarrow e_2] \longrightarrow d}{\mathbf{apply}(e_1, e_2) \longrightarrow d}$$

$$\frac{e \longrightarrow \mathbf{constr}(i, e') \qquad \mathbf{lookup}(t, i, e'') \qquad \mathbf{apply}(e'', e') \longrightarrow d}{\mathbf{case}(e, t) \longrightarrow d}$$

$$\frac{e \longrightarrow \mathbf{struct}(t) \qquad \mathbf{lookup}(t, i, e') \qquad e' \longrightarrow d}{\mathbf{proj}(e, i) \longrightarrow d}$$

$$\frac{e[i \leftarrow \mathbf{rec}(i, e)] \longrightarrow d}{\mathbf{rec}(i, e) \longrightarrow d}$$

$$\mathbf{lambda}(i, e) \longrightarrow \mathbf{lambda}(i, e)$$

$$\mathbf{constr}(i, e) \longrightarrow \mathbf{constr}(i, e)$$

$$\mathbf{struct}(t) \longrightarrow \mathbf{struct}(t)$$

Svar: Lösningen finns i filen `isnat.html` under kursens hemsida

Lycka till!