Tik-79.148 **Kevät 2000**
Tietojenkäsittelyteorian perusteet
Laskuharjoitus 10
Ratkaisut

1. Jos haluamme todistaa, että jokin ongelma on ratkeamaton, voimme joko

   (i) Lähteä liikkeelle oletuksesta, että ongelma on ratkeava, ja johtaa siitä ristiriita käyttäen valitsemaamme formalismia.

   (ii) Jos jo tiedämme jonkin toisen kielen olevan ratkeamaton, voimme yrittää palauttaa (redusoida) oman ongelmamme tähän ongelmaan. Tähänkin on kaksi eri vaihtoehtoa. Ensimmäinen vaihtoehto on se, että osoitamme, että mikäli pystyisimme ratkaisemaan tutkimamme ongelman voisimme myös käyttää sitä kyseisen ratkeamattoman ongelman ratkaisemiseen. Toinen mahdollinen tapa on osoittaa, että pystyäksemme ratkaisemaan ongelmamme meidän täytyisi pystyä ratkaisemaan ratkeamaton ongelma.

   Tässä vastauksessa käytetään jälkimmäistä tapaa. Todistuksen ideana on osoittaa, että pystyäksemme ratkaisemaan ongelman "pysähtyykö mielivaltainen Turingin kone M tyhjällä syötteellä", meidän täytyisi pystyä ratkaisemaan ongelma "pysähtyykö mielivaltainen Turingin kone M' syötteellä x". Jälkimmäinen ongelma on Turingin koneen yleinen pysähtymisongelma, joka on oppikirjassa todistettu ratkeamattomaksi.

   Mikäli ongelma on ratkeava, on olemassa Turingin kone, joka ratkaisee kielen

   $$L = \{M \mid M \text{ pysähtyy syötteellä } e\} \ .$$

   Toisin sanoen, on olemassa Turingin kone, joka saadessaan syötteekseen minkä tahansa toisen Turingin koneen kuvauksen pystyy selvittämään pysähtyykö tämä kone tyhjällä syötteellä. Todistukseksi kielen ratkeamattomuudesta riittää, että löydämme yhdenkin Turingin koneen, jonka pysähtymistä ei pystytä selvittämään.

   Muodostetaan Turingin kone M, joka toimii seuraavasti: saatuaan syötteekseen tyhjän merkkijonon kone kirjoittaa aluksi nauhalle mielivaltaisen merkkijonon x (kirjoitettava merkkinono voidaan valita epädeterministisesti). Tämän jälkeen kone siirtyy simuloimaan mielivaltaista Turingin konetta M', eli se tekee tismalleen samat siirtymät kuin kone M':kin tekisi.

   Nyt kone M pysähtyy syötteellä e täsmälleen silloin , kun kone M' pysähtyy syötteellä x. Koska jälkimmäinen ongelma ei ole ratkeava, ei myöskään voi olla olemassa mitään algoritmia sen selvittämiseksi, pysähtyykö M syötteellä e. Näin ollen ongelma on ratkeamaton.

2. Olkoon annettuina kaksi Turing-hyväksyttävää kieltä L_1 ja L_2. Tällöin on määritelmän mukaan olemassa kaksi Turingin konetta M_1 ja M_2, jotka hyväksyvät kyseiset kielet (eli $L(M_1) = L_1$ ja $L(M_2) = L_2$). Muodostetaan näitä käyttäen 2-nauhaiset Turingin koneet $M_\cap$ ja $M_\cup$, jotka hyväksyvät kielten leikkauksen ja unionin.

(a) Leikkaus:

Muodostetaan $M_\cap$ siten, että suorituksen aluksi $M_\cap$ kopio syötteen w 2-nauhalle. Tämän jälkeen M_cap simuloi koneen M_1 toimintaa syötteellä w käyttäen ensimmäistä nauhaa työnauhanaan. Mikäli M_1 pysähtyy, tiedetään, että $w \in L_1$. Nyt voidaan simuloida konetta M_2 syötteellä w käyttäen toista nauhaa työnauhana. Mikäli M_2:kin pysähtyy, kuuluu sana molempiin kieliin, ja $M_\cap$ hyväksyy sanan. Mikäli w ei kuulu jompaan kumpaan kieleen, jää kyseisen kielen tunnistava kone ikuiseen silmukkaan, eikä kone $M_\cap$ pysähdy. Näin ollen $M_\cap$ hyväksyy kielen $L_1 \cap L_2$.

(b) Unioni:

Kielten unionin tunnistava kone $M_\cup$ muodostetaan samalla periaatteella. Tässä täytyy kuitenkin huomata, että koneita M_1 ja M_2 ei voida simuloida peräkkäin, koska M_1:n jäädessä ikuiseen silmukkaan (kun $w \notin L_1$) ei päästä koskaan tarkistamaan kuuluisiko sana w kieleen L_2.

Ratkaisuna on simuloida koneita M_1 ja M_2 rinnakkain, yksi askel kerrallaan. Ensiksi suoritetaan koneen M_1 laskennan ensimmäinen siirtymä, sitten M_2:n ensimmäinen siirtymä, sitten M_1:n toinen siirtymä ja niin edespäin. Mikäli sana kuuluu jompaan kumpaan kieleen, joko M_1 tai M_2 pysähtyy joskus, jolloin $M_\cup$ hyväksyy syötteen.

3. Primitiivirekursiiviset funktiot muodostetaan kolmesta perusfunktiosta käyttäen kahta yhdistämissääntöä.

Perusfunktioita ovat

(a) *Nollafunktiot* $\mathsf{zero}_k(n_1, \ldots, n_k) = 0$ kaikille $k, n_1, \ldots, n_k \in \mathbb{N}$.

(b) *Identiteettifunktio* $\mathsf{id}_{k,j}(n_1, \ldots, n_k) = n_j$, kaikille $k \geq j > 0$, $n_1, \ldots, n_k \in \mathbb{N}$.

(c) *Seuraajafunktio* $\mathsf{succ}(n) = n + 1$ kaikille $n \in \mathbb{N}$.

Perusfunktioista zero ja succ määrittelevät luonnolliset luvut:

$$\begin{aligned} \mathsf{zero}() &= 0 \\ \mathsf{succ}(\mathsf{zero}()) &= 1 \\ \mathsf{succ}(\mathsf{succ}(\mathsf{zero}())) &= 2 \\ &\vdots \\ \mathsf{succ}(n) &= n + 1 \end{aligned}$$

Perusfunktioita voidaan yhdistää käyttäen kompositiota ja primitiivirekursiota.

(a) Olkoon $k, l \geq 0$, g k-paikkainen funktio ($g : \mathbb{N}^k \to \mathbb{N}$) ja $h_1, \ldots, h_k$ joukko l-paikkaisia funktioita. Nyt g:n *kompositio* $h_1, \ldots, h_k$:n kanssa on l-paikkainen funktio

$$f(n_1, \ldots, n_l) = g(h_1(n_1, \ldots, n_l), \ldots, h_k(n_1, \ldots, n_l))$$

Kompositiossa annetaan yhden funktion palauttama arvo toisen funktion argumentiksi. Kaikkien h_i-funktioiden ei tarvitse olla erilaisia. Esimerkiksi funktio $f(n, m) = n^2 + m^2$ saadaan muodostettua kompositiolla funktioista plus ja times seuraavasti:

$$f(n, m) = \mathsf{plus}(\mathsf{times}(n, n), \mathsf{times}(m, m))$$

Tässä $l = k = 2$, $g = \mathsf{plus}$ ja $h_1 = h_2 = \mathsf{times}$.

(b) Kaikille $k \geq 0$, olkoon g k-paikkainen funktio ja h $k+2$-paikkainen funktio. Näistä saadaan muodostettua *primitiivirekursiolla* $k+1$-paikkainen funktio f seuraavasti:

$$\begin{aligned} f(n_1, \ldots, n_k, 0) &= g(n_1, \ldots, n_k) \\ f(n_1, \ldots, n_k, m+1) &= h(n_1, \ldots, n_k, m, f(n_1, \ldots, n_k, m)) \end{aligned}$$

kaikille $n_1, \ldots, n_k, m \in \mathbb{N}$.

Primitiivirekursiossa on yksinkertainen perustapaus ($m = 0$), sekä joukko rekursiokaavoja, joita käyttämällä saadaan funktion arvon laskeminen palautettua perustapaukseen.

Primitiivirekursiivisia funktioita ovat kaikki perusfunktiot sekä niistä kompositiolla ja primitiivirekursiolla saatavat funktiot.

Tehtävässä piti todistaa, että funktio

$$f(n) = \text{"}n + 1\text{:s pariton luonnollinen luku"}$$

on primitiivirekursiivinen. Funktio voidaan palauttaa perusfunktioihin muutamallakin eri tavalla, tässä yksi niistä:

$$\begin{aligned} f(0) &= 1 \\ f(m+1) &= f(m) + 2 \end{aligned}$$

Koska luvut 1 ja 2 eivät ole perusfunktioita, korvataan ne vielä käyttämällä succ-funktion kompositiota:

$$\begin{aligned} f(0) &= \mathsf{succ}(\mathsf{zero}()) \\ f(m+1) &= \mathsf{succ}(\mathsf{succ}(f(m))) \end{aligned}$$

Tässä $k = 0$, $g = \mathsf{succ}(\mathsf{zero}())$ ja $h = \mathsf{succ}(\mathsf{succ}(f(m)))$.

4. Lisäämällä primitivirekursiivisten funktioiden määritelmään uusi operaatio, *minimointi*, saadaan muodostettua laajempia funktioluokkia. Minimointilause on muotoa

$$\mu.z[P(z)] \ ,$$

missä z on muuttuja ja $P(z)$ on jokin predikaatti (eli funktio, jolla on kaksi mahdollista arvoa: *tosi* ja *epätosi*), jonka arvo riippuu muuttujasta z. Minimointioperaattorin arvo on pienin mahdollinen z:n arvo, jolla predikaatti $P(z)$ on tosi.

Formaalimmin: Olkoon g funktio, jolla on $k+1$ argumenttia, $k \geq 0$. Tällöin g:n k-argumenttinen minimointifunktio f on määritelty seuraavasti:

$$f(n_1, \ldots, n_k) = \begin{cases} \text{pienin } m, \text{ jolla } g(n_1, \ldots, n_k, m) = 1 & \text{, jos } m \text{ on olemassa} \\ 0 & \text{, muulloin} \end{cases}$$

Predikaattia g kutsutaan *säännölliseksi*, jos ehdon toteuttava m on olemassa kaikille mahdollisille g:n argumenteille. Yleisessä tapauksessa on mahdotonta sanoa onko g säännöllinen vai ei, joten predikaatin minimiä ei välttämättä voida laskea. Ongelma on pohjimmiltaan sama kuin Turingin koneen pysähtymisongelma.

Funktiota kutsutaan μ-rekursiiviseksi (tai pelkästään rekursiiviseksi), mikäli se saadaan muodostettua perusfunktioista käyttäen kompositiota, primitiivirekursiota ja säännöllisten predikaattien minimointia.

Tavallisen minimoinnin lisäksi voidaan käyttää myös *rajoitettua minimointia*. Rajoitettu minimointi on muotoa:

$$\mu.z_{\leq x}[P(z)] \ ,$$

missä x on z:n suurin mahdollinen arvo. Mikäli mikään $z \leq x$ ei toteuta predikaattia $P(z)$, asetetaan minimoinnin tulokseksi 0. Rajoitetun minimoinnin arvo voidaan aina laskea, koska x asettaa ylärajan mahdollisten laskenta-askelien määrälle. Rajoitettu minimointi voidaan toteuttaa primitiivirekursiolla, joten primitiivirekursiivisten funktioiden luokka on suljettu sen suhteen.

Intuitiivisesti rajoitettu minimointi voitaisiin kuvata ohjelmallisesti rajoitetulla for-silmukalla:

for i := 1 to n do

Oleellista silmukassa on se, että tarvittavien iteraatiokertojen lukumäärälle voidaan esittää jokin yläraja. Yksinkertaisena nyrkkisääntönä voidaan pitää, että funktio on primitiivirekursiivinen, jos sen voi toteuttaa joukolla sisäkkäisiä for-silmukoita.

Vastaavasti rajoittamaton minimointi vastaisi ohjelmarakennetta

while not end do

Tässä tapauksessa oleellista on se, että ei ole mahdollista esittää etukäteen jotain tiettyä ylärajaa silmukan suorittamiskertojen määrälle, vaan silmukassa pysytään niin kauan, kunnes lopetusehto toteutuu.

Funktio on μ-rekursiivinen, jos se voidaan laskea joukolla sisäkkäisiä **while**-silmukoita, joiden lopetusehdot tulevat aina lopulta voimaan.

Kahden kokonaisluvun jakojäännös voidaan laskea seuraavalla μ-rekursiivisella funktiolla:

$$\text{mod}\ (x, y) = \begin{cases} 0 & , x = 0 \vee y = 0 \\ \mu.z_{\leq x}\left[\mu.w_{\leq x}\left[y \cdot w + z = x\right]\right] & \text{, muulloin.} \end{cases}$$

Itseasiassa ylläoleva funktio on myös primitiivirekursiivinen, koska kummatkin minimoinnit ovat rajoitettuja.

Funktio vastaa seuraavanlaista C-kielen ohjelmaa:

```
int modulo(int x, int y)
{
    int z, w;
    if (x == 0 || y == 0)
        return 0;

    for (z = 0; z <= x; z++) {
        for (w = 0; w <= x; w++) {
            if (w*y + z == x)
                return z;
        }
    }
}
```