Tik-79.148 **Kevät 1999**
Tietojenkäsittelyteorian perusteet
Laskuharjoitus 11
Ratkaisut

1. Primitiivirekursiiviset funktiot muodostetaan kolmesta perusfunktiosta ($\mathbb{N}^k \to \mathbb{N}$) 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:

$$f(n_1, \ldots, n_k, 0) = g(n_1, \ldots, n_k)$$
$$f(n_1, \ldots, n_k, m+1) = h(n_1, \ldots, n_k, m, f(n_1, \ldots, n_k, m))$$

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.

Intuitiivisesti primitiivirekursiivisiin funktioihin kuuluvat kaikki ne funktiot, jotka voitaisiin laskea ohjelmallisesti rajoitetulla for-silmukalla:

for i := 1 to n do

Oleellista silmukassa on se, että tarvittavien iteraatiokertojen lukumäärälle voidaan esittää jokin yläraja.

Vastaavasti μ-rekursiivisiin funktioihin (jotka on määritelty kirjassa ja luennolla) kuuluvat primitiivirekursiivisten funktioiden lisäksi ne funktiot, joiden ohjelmallinen toteutus on muotoa:

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.

Tarkkaan ottaen μ-rekursiivisiin funktioihin kuuluvat ainoastaan ne funktiot, joilla silmukan suoritus joskus pysähtyy. Samoin kuin on mahdotonta sanoa yleisessä tapauksessa pysähtyykö jokin mielivaltainen Turingin-kone tietyllä syötteellä, on myös yleisessä tapauksessa mahdotonta sanoa onko jokin funktio μ-rekursiivinen.

Tehtävässä piti todistaa, että funktio $f(n) =$ “ $n+1$:s pariton luonnollinen luku” on primitiivirekursiivinen. Funktio voidaan palauttaa perusfunktioihin muutamallakin eri tavalla, tässä yksi niistä:

$$f(0) = 1$$
$$f(m+1) = f(m) + 2$$

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 $h = \mathsf{succ}(\mathsf{succ}(f(m)))$.

2. Koska rekursiiviset funktiot saavat argumenteikseen luonnollisia lukuja, niillä ei voi suoraan käsitellä merkkijonoja. Mikäli näin kuitenkin halutaan tehdä, täytyy merkkijonot koodata numeroiksi systemaattisella tavalla. Tällaista koodausta kutsutaan *Gödel-numeroinniksi* (loogikko Kurt Gödelin 1906-1978 mukaan). On olemassa monia vaihtoehtoisia tapoja määritellä Gödel-numerointi. Kirjassa koodataan n-kirjaimisen aakkoston merkkijonot $n+1$-kantaiseksi kokonaisluvuiksi, jotka edelleen muutetaan kymmenjärjestelmän luvuiksi.

   Tehtävän aakkoston $\Delta = \{a, b, c\}$ symbolit asetetaan järjestykseen

   $$\begin{aligned} d_1 &= a \\ d_2 &= b \\ d_3 &= c \end{aligned}$$

   Gödel-numeroinnin kantaluvuksi saadaan $\beta = |\Delta| + 1 = 4$.

   Merkkijono $w = d_{i_1} d_{i_2} \ldots d_{i_k}$ muutetaan numeroksi käyttäen tavallista tapaa lukujärjestelmän vaihtamiseksi:

   $$\mathsf{gn}(w) = \beta^{k-1} \cdot i_1 + \beta^{k-2} \cdot i_2 + \cdots + \beta^1 \cdot i_{k1} + i_k$$

   Oppikirjassa määritellään lisäksi symboli d_0, joka vastaa tyhjää merkkijonoa e. Symboli on lisätty sen vuoksi, että muussa tapauksessa olisi olemassa lukuja (kuten β), joita ei vastaisi mikään merkkijono. Tyhjän merkkijonon lisääminen puolestaan aiheuttaa sen, että yhdellä merkkijonolla on monta Gödel-numeroa, jotka saadaan lisäämällä jonon symbolien väliin tyhjiä merkkejä. Tällöin merkkijonon varsinainen Gödel-numero on se, jossa ei ole yhtään tyhjää merkkiä.

   (a) Merkkijonon abc Gödel-numero löydetään seuraavasti.

   $$\begin{aligned} \mathsf{gn}(abc) &= 1 \cdot 4^2 + 2 \cdot 4^1 + 3 \cdot 4^0 \\ &= 27 \end{aligned}$$

(b) Gödel-numeroa 19 vastaava merkkijono on:

$$19 = 1 \cdot 4^2 + 0 \cdot 4^1 + 3 \cdot 4^0$$
$$\mathsf{gn}^{-}1(19) = d_1 d_0 d_3 = aec = ac$$

Gödel-numeroinnin käytännön merkitys on siinä, että niiden avulla voidaan määritellä funktioita, jotka käsittelevät toisia funktioita. Tällöin funktio saa argumentikseen toista funktiota vastaavan Gödel-numeron.

Gödel itse käytti numerointiaan kuuluisassa "epätäydellisyystodistuksessaan", jossa hän osoitti, että millä tahansa riittävän ilmaisuvoimaisella[1] formalismilla voidaan esittää väitteitä, joiden totuusarvoa ei pystytä selvittämään. Tällä kurssilla esiintulleista formalismeista "riittävän ilmaisuvoimaisia" ovat Turingin-koneet (Turingin-koneen pysähtymisongelma), μ-rekursiiviset funktiot (löytääkö rajoittamaton minimointi vastauksen) sekä vapaat kieliopit (onko $w \in L(G)$).

3. 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 $L = \{M \,|\, M \text{ pysähtyy syötteellä } e\}$. Toisin sanoen, on olemassa Turingin-kone, joka saadessaan syötteekseen minkä tahansa toisen

[1]Kieli on "riittävän ilmaisuvoimainen", mikäli kielellä pystytään esittämään luonnollisten lukujen numeroteoria.

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 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.