Tik-79.148 **Kevät 2001**
Tietojenkäsittelyteorian perusteet
Laskuharjoitus 11
Ratkaisut

1. 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:

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

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

$$f(0) = \mathsf{succ}(\mathsf{zero}())$$
$$f(m+1) = \mathsf{succ}(\mathsf{succ}(f(m)))$$

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

2. 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:

$$\mathsf{for}\ \mathsf{i} := 1\ \mathsf{to}\ n\ \mathsf{do}$$

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

$$\mathbf{while}\ \mathsf{not\ end\ do}$$

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:

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

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;
        }
    }
}
```

3. 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:

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

Gödel-numeroinnin käytännön merkitys on siinä, että sen 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 tyypin 0 kieliopit (voidaanko sana w johtaa).

4. Ongelmaluokkaan NP kuuluvat kaikki ne ongelmat, jotka voidaan ratkaista epädeterministisellä Turingin koneella polynomisessa ajassa syötteen

[1] Formalismi on "riittävän ilmaisuvoimainen", mikäli kielellä pystytään esittämään luonnollisten lukujen teoria.

pituuteen. Käytännössä tämä tarkoittaa sitä, että kaikki ongelmat, joiden vastauksen oikeellisuus voidaan tarkistaa polynomisessa ajassa, ovat NP-luokassa.

Ongelma on NP-täydellinen, mikäli kaikki NP-luokan ongelmat voidaan palauttaa siihen polynomisessa ajassa. Tässä perusajatuksena on osoittaa, että ongelma A on vähintään yhtä vaikea ratkaistavaksi kuin ongelma B. Tämä tehdään näyttämällä, miten ongelma B voidaan muuttaa ongelmaksi A. Muunnoksesta käytetään termiä *reduktio*.

Ongelman todistaminen NP-täydelliseksi tapahtuu yleensä kahdessa vaiheessa:

(a) Todistetaan, että ongelma kuuluu luokkaan NP.

(b) Osoitetaan, että jokin NP-täydellinen ongelma voidaan redusoida tutkittavaan ongelmaan.[2]

Graafin ydin on joukko solmuja, joille pätevät seuraavat ehdot:

(a) Ytimeen kuuluvien solmujen välillä ei ole kaaria.

(b) Kaikkiin ytimeen kuulumattomiin solmuihin päästään yhdellä askeleella jostain ytimeen kuuluvasta solmusta.

Graafin ytimen löytämisongelma kuuluu selvästikin luokkaan NP, koska se voidaan ratkaista epädeterministisellä Turingin koneella seuraavasti:

(a) Arvataan solmujoukko K. Tähän kuluu $O(|K|)$ askelta.

(b) Käydään läpi kaikki K:n solmuista lähtevät kaaret (a, b). Mikäli kaari menee johonkin toiseen K:hon kuuluvaan solmuun, hylätään vastaus. Muussa tapauksessa merkitään solmu b tarkistetuksi. Tähän kuluu $O(|E|)$ askelta.

(c) Lopuksi tarkistetaan, että kaikki ytimeen kuulumattomat solmut ovat merkittyjä. Tämä voidaan tehdä ajassa $O(|V|)$.

Seuraavaksi täytyy redusoida jokin NP-täydellinen ongelma peittävän joukon etsimisongelmaan. Yksi perustavinta laatua olevista NP-täydellisistä ongelmista on niin kutsuttu 3-SAT-ongelma. Ongelmassa on annettuna joukko lauselogiikan klausuuleita, joissa kaikissa on täsmälleen kolme literaalia, ja tarkoituksena on löytää jokin tapa asettaa loogisille muuttujille totuusarvot siten, että kaikissa klausuuleissa on vähintään yksi tosi literaali.

Esimerkiksi klausuulijoukko:

$$\{\{x_1, \neg x_2, x_3\}, \{\neg x_1, \neg x_2, \neg x_3\}\}$$

on toteutuva, koska valitsemalla x_1:n ja x_2:n arvoiksi *tosi* sekä x_3:n arvoksi *epätosi*, on kummassakin klausuulissa on vähintään yksi tosi literaali. Muuttuja x_1 tekee ensimmäisen klausuulin todeksi ja muuttuja x_3

[2] Tarkkaavainen lukija voi tässä kohtaa ihmetellä, miten ensimmäinen ongelma todistettiin NP-täydelliseksi. Tämä tehtiin simuloimalla suoraan deterministisen Turingin koneen laskentaa. Useimmiten on huomattavasti helpompaa käyttää jotain NP-täydelliseksi ongelmaksi tunnettua ongelmaa todistuksen tekemiseen.

tekee toisen klausuulin todeksi ($\neg x$ on tosi täsmälleen silloin, kun x ei ole tosi). Tällaista muuttujien arvojen valintaa kutsutaan *valuaatioksi*, ja siitä käytetään merkintää $\mathcal{V} = \{x_1, x_2, \neg x_3\}$, tai $\mathcal{V}(x_1) = \mathcal{V}(x_2) = T$, $\mathcal{V}(x_3) = F$.

3-SAT-ongelma voidaan muuntaa graafin peittämisongelmaksi seuraavalla tavalla:

Olkoon 3-SAT-ongelmassa esiintyvien muuttujien joukko $x_1, \ldots, x_n$ ja klausuulien joukko $C = \{C_1, \ldots, C_m\}$ Muodostetaan graafi $G = (V, E)$ siten, että jokaista klausuulia C_i kohden verkkoon tulee yksittäinen solmu c_i ja jokaista muuttujaa x_i kohden tulee kaksi solmua: x_i ja $\neg x_i$. Graafin kaarirelaatio E määritellään seuraavasti:

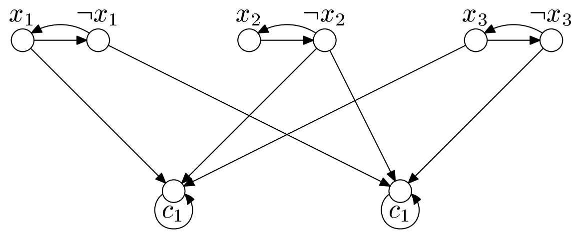
(a) Kaikilla x_i, kaaret $(x_i, \neg x_i), (\neg x_i, x_i) \in E$.

(b) Kaikilla $x_i \in C_j$, $(x_i, c_j) \in E$.

(c) Kaikilla $\neg x_i \in C_j$, $(\neg x_i, c_j) \in E$.

(d) Kaikilla c_i, $(c_i, c_i) \in E$.

Esimerkiksi ylläolevaa klausuulijoukkoa vastaava verkko on:



Graafin rakenne on suunniteltu siten, että solmu x_i kuuluu ytimeen, jos ja vain jos muuttuja x_i on tosi valuaatiossa, joka toteuttaa klausuulijoukon. Samoin solmu $\neg x_i$ kuuluu ytimeen, mikäli x_i ei ole tosi toteuttavassa valuaatiossa. Solmujen x_i ja $\neg x_i$ välinen keskinäinen poissulkevuus toteutetaan sillä, että niiden välillä on kaaret kumpaankin suuntaan, joten niiden molempien mukaanottaminen rikkoisi ytimen määritelmän. Toisaalta, jomman kumman solmuista täytyy kuulua ytimeen, sillä niihin ei tule mitään muita kaaria, eikä määritelmän toinen ehto voi muutoin toteutua. Mitään muita solmuja ei ytimeen voi kuulua, sillä kaikista solmuista c_i on kaari takaisin itseensä.

Ylläolevassa verkossa on ydin $K = \{x_1, x_2, \neg x_3\}$, joka vastaa valuaatiota $x_1 = T$, $x_2 = T$, $x_3 = F$.

Todistetaan vielä formaalisti, että graafilla on ydin tismalleen silloin, kun klausuulijoukko on toteutuva.

Mikäli graafilla G on ydin K, voidaan klausuulijoukon toteuttava valuaatio muodostaa seuraavasti: $\mathcal{V}(x_i) = T$, joss $x_i \in K$, muuten $\mathcal{V}(x_i) = F$. Koska K on ydin, on jokaista c_i:tä kohden K:ssa ainakin yksi solmu x siten, että $(x, c_i) \in E$. Graafin rakenteen perusteella $x \in C_i$, joten klausuuli C_i toteutuu. Koska sama pätee kaikille klausuuleille, toteuttaa valuaatio $\mathcal{V}$ klausuulijoukon.

Mikäli klausuulijoukko C on toteutuva, sillä on ainakin yksi valuaatio $\mathcal{V}$, joka toteuttaa sen. Muodostetaan K siten, että $x_i \in K$, jos $\mathcal{V}(x_i) = T$, ja $\neg x_i \in K$, jos $\mathcal{V}(x_i) = F$. Koska kaikille klausuuleille löytyy ainakin yksi literaali x, joka on tosi, on kaikilla solmuilla c_i graafin rakenteen perusteella vähintään yksi ytimeen kuuluva edeltäjä. Valuaatiossa on kullakin loogisella muuttujalla tismalleen yksi arvo, joten K:ssa ei voi olla kahta solmua, joiden välillä on kaari. Näin ollen K on ydin.