TAMPEREEN YLIOPISTO
Pro gradu-tutkielma

Juho Piirto

Laskettavuuden teorian perusteet ja Churchin teesi

Matematiikan, tilastotieteen ja filosofian laitos
Matematiikka
2007

Tampereen yliopisto
Matematiikan, tilastotieteen ja filosofian laitos

PIIRTO JUHO: Laskettavuuden teorian perusteet ja Churchin teesi

Pro gradu-tutkielma, 40s.
Matematiikka
Huhtikuu 2007

TIIVISTELMÄ

Tämän työn aiheena on laskettavuuden teorian perusteet. Tarkoituksenani on ensinnäkin pohtia hieman mitä tarkoitetaan "laskemisella" ja kuinka laskemiseksi luokiteltava prosessi on formalisoitavissa. Esittelen rajoittamattoman rekisterikoneen, osittainrekursiiviset funktiot ja Turingin koneen, jotka ovat vaihtoehtoisia tapoja formalisoida laskeminen. Työn lopullisena tavoitteena on osoittaa, että näiden kolmen järjestelmän kautta määriteltyjen laskettavien funktioiden joukot ovat yhtäpitäviä, ja yleistää tämä tulos niinsanottuun *Churchin teesiin*, jonka mukaan kaikki mahdolliset laskettavuuden käsitteen formaalit määritelmät antavat yhtäpitävät laskettavien funktioiden joukot.

Päälähteenäni olen käyttänyt Nigel Cutlandin kirjaa Computability [1], lukuunottamatta lukua 4, jossa olen käyttänyt Piergiorgio Odifreddin kirjaa Classical Recursion Theory [2].

Sisältö

Johdanto

Hyödyllisiä käsitteitä

Tämä tutkielma ei edellytä lukijalta matematiikan perustietojen lisäksi mitään erityisiä esitietoja aiheesta. On kuitenkin tarpeellista määritellä aluksi muutamia tässä tutkielmassa käytettyjä merkintöjä:

"Rekisterillä" tarkoitetaan yleensä rekisterin arvoa. "Funktio" taas ei ole välttämättä määritelty koko määrittelyjoukossaan ja määrittelyjoukossaan kaikkialla määritelty funktiota sanotaan täydelliseksi funktioksi. "$Dom(f)$" tarkoittaa funktion f määrittelyjoukkoa. Merkintä $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ tarkoittaa n-paikkaista vektoria, jonka muuttujina ovat $x_1, x_2, \ldots, x_n$. Merkintä $\alpha(x) \simeq \beta(x)$ tarkoittaa, että kaikilla x, merkinnät $\alpha(x)$ ja $\beta(x)$ ovat molemmat joko määriteltyjä tai molemmat ovat määrittelemättömiä ja jos ne ovat määriteltyjä, niin niiden arvot ovat yhtäsuuria. Todistuksissa käytetty lyhenne "joss" tarkoittaa samaa kuin "jos ja vain jos". Operaatio $*$ on "suljettu" joukon G suhteen, jos $\forall a, b \in G : a * b \in G$.

Algorimi ja laskettavuuden intuitiivinen käsite

Kun aikuinen ihminen laskee jotakin melko yksinkertaista laskua, kuten 5+7, ei ole luultavaa, että hän mitenkään erityisemmin miettisi päässään tapahtuvaa laskuoperaatiota. Numero 12 kuin "ilmestyy" mieleen laskua silmäillessä. Pienelle lapselle tilanne ei kuitenkaan ole näin yksinkertainen. Hän voi laskiessaan joutua käyttämään sormiaan apunaan ja mahdollisesti jopa varpaitaan laskun summan kasvaessa kymmentä suuremmaksi. Tällöin on mahdollista, että lapsi laskee ensin toiseen käteensä viisi sormea pystyyn ja lähtee siihen lisäämään seitsemää sormea. Kaikkien kymmenen sormen ollessa pystyssä hän saattaa laittaa mieleensä luvun kymmenen, laittaa molemmat kädet takaisin nyrkkiin ja jatkaa laskua lisäämällä mielessä olevaan kymmeneen jäljellä olevat kaksi sormea. Tämän kaltainen tekniikka on sovellettavissa muillekkin luvuille kuin viisi ja seitsemän. Edellä mainitusta, hieman helmitaululla laskemista muistuttavasta, tavasta saadaan yleistämällä yksikäsitteinen ohje lukujen x ja y yhteen laskemiseksi. Tällaisia ohjeita tai ohjejoukkoja kutsutaan *algoritmeiksi*. Eräs algoritmin täsmällinen määritelmistä on J. G. Brookshearin seuraavalla tavalla esittämä määritelmä: "Tarkasti ottaen algoritmi on äärellinen joukko täsmällisiä, suoritettavissa olevia ohjeita, jotka ohjaavat päättyvän tehtävän suoritusta."

Edellä on siis annettu algoritmi yhteenlaskua varten ja asiaa hieman tarkastelemalla osoittautuu, että muutkin yksinkertaiset laskuoperaatiot kuten vähennyslasku ja kertolasku ovat algoritmisoitavissa vastaavan kaltaisil-

la tavoilla. Huomaamme, että ei ole kuitenkaan kovin hyödyllistä kehittää yhteenlaskulle omaa algoritmiaan, kertolaskulle omaansa ja niin edelleenja näin määritellä rajat sille, mitä voimme laskea. Paljon hyödyllisempää olisi, jos meillä olisi jokin järjestelmä, josta erityistapauksina saataisiin halutut laskut. Ajatusta hieman jatkamalla herää kysymys, että minkälainen tämä järjestelmä olisi ja mitkä kaikki laskuoperaatiot voitaisiin algoritmisoida?

Tämä kysymys on tärkeä esimerkiksi tietotekniikan kannalta, sillä jos meillä ei olisi mitään keinoa formalisoida laskuoperaatoitamme, ei meillä voisi olla mitään keinoa valmistaa digitaalisia tietokoneita tai edes yksinkertaisia taskulaskimia. Tätä ongelmaa tutkimalla pystymme lisäksi tarkastelemaan, mikä tietokoneilla voidaan teoriassa laskea.

Edellä mainittu kysymys laskemisen formalisoinnista heräsi 1900-luvun alkupuoliskolla, jolloin mm. Alan Turing ja Alonzo Church tutkivat asiaa ja loivat toisistaan riippumatta omat formaalin laskemisen järjestelmänsä. Tällaisia järjeselmiä on nykyään olemassa useita erilaisia ja aloitamme laskettavuuden käsitteen tutkimisen tarkastelemalla kolmea niistä. Tämän jälkeen osoitamme, että kaikki esitellyt tavat mekanisoida laskeminen antavat samat laskettavien funktioiden joukot. Tämä tarkoittaa sitä, että mikäli voimme laskea jonkun laskun jonkun esitellyn järjestelmän avulla, voimme laskea sen minkä tahansa muun järjestelmänkin avulla.

1 Rajoittamaton rekisterikone

Rajoittamaton rekisterikone (unlimited register machine) on eräs laskennan teorian malli. Sen ensimmäisen, tässä esitetystä mallista poikkeavan, version esittivät Shepherdson ja Strugis vuonna 1963. Seuraavaksi käymme läpi kuinka rajoittamaton rekisterikone (RRK tästä eteenpäin) toimii ja myöhemmin tarkastelemme, mitä sillä voi tehdä.

RRK:ssa on ääretön määrä *rekistereitä* (merk. $R_1, R_2, R_3, \ldots$), joista jokainen sisältää yhden luonnollisen luvun. Rekisterissä R_n olevaa lukua merkitään r_n:llä. Rekisterikonetta voidaan kuvata seuraavalla tavalla:

R_1	R_2	R_3	R_4	R_5	R_6	$\ldots$
4	2	1	3	0	0	$\ldots$

Edellisessä kuvaajassa rekisterin R_1 arvo on siis 4, rekisterin R_2 arvo on 2 ja niin edelleen. Täten siis $r_1 = 4$, $r_2 = 2$, $\ldots$

Rekisterikoneen sisältämiä lukuja voidaan muuttaan koneen tunnistamilla erityisillä *operaatioilla*. Nämä operaatiot ovat yksinkertaisia laskuoperaatoita. Äärellinen lista tällaisia operaatioita muodostaa *ohjelman*. Näitä operaatioita on olemassa neljä kappaletta: nollaoperaatio, seuraajaoperaatio, muunto-operaatio ja hyppyoperaatio.

Määritelmä 1 (**Nollaoperaatio**). *Jokaiselle* $n = 1, 2, 3 \ldots$ *on olemassa* nollaoperaatio $Z(n)$. *Nollaoperaatio* $Z(n)$ *muuntaa luvun* r_n *nollaksi ja jättää muut rekisterit ennalleen. Nollaoperaatiota* $Z(n)$ *merkitään myös* $0 \to R_n$ *tai* $r_n := 0$.

Esimerkki 1. Olkoon RRK seuraavassa tilassa:

R_1	R_2	R_3	R_4	R_5	R_6	$\ldots$
2	6	1	6	0	0	$\ldots$

Tällöin operaatio $Z(2)$ muuntaa RRK:n seuraavanlaiseksi:

R_1	R_2	R_3	R_4	R_5	R_6	$\ldots$
2	0	1	6	0	0	$\ldots$

Määritelmä 2 (**Seuraajaoperaatio**). *Jokaiselle* $n = 1, 2, 3, \ldots$ *on olemassa* seuraajaoperaatio $S(n)$. *Seuraajaoperaatio* $S(n)$ *muuntaa luvun* r_n *luvuksi* r_1+1 *ja jättää muut rekisterit ennalleen. Seuraajaoperaatiota* $S(n)$ *merkitään myös* $r_n + 1 \to R_n$ *tai* $r_n := r_n + 1$.

Esimerkki 2. Olkoon RRK tilassa

R_1	R_2	R_3	R_4	R_5	R_6	$\ldots$
2	0	1	6	0	0	$\ldots$

Tällöin komento $S(3)$ muuntaa koneen tilaan

R_1	R_2	R_3	R_4	R_5	R_6	$\ldots$
2	0	2	6	0	0	$\ldots$

Määritelmä 3 (**Muunto-operaatio**). *Jokaiselle* $m = 1, 2, 3, \ldots$ *ja* $n = 1, 2, 3, \ldots$ *on olemassa* muunto-operaatio $T(m, n)$. *Muunto-operaatio* $T(m, n)$ *vaihtaa luvun* r_n *luvuksi* r_m *jättäen muut rekisterit (myös* R_m*:n) ennalleen. Muunto-operaatiota* $T(m, n)$ *merkitään myös* $r_m \to R_n$ *tai* $r_n := r_m$.

Esimerkki 3. Olkoon RRK tilassa

R_1	R_2	R_3	R_4	R_5	R_6	$\ldots$
2	0	2	6	0	0	$\ldots$

Tällöin muunto-operaatio $T(4, 3)$ muuttaa RRK:n tilaan

R_1	R_2	R_3	R_4	R_5	R_6	$\ldots$
2	0	6	6	0	0	$\ldots$

Nolla-, seuraaja- ja muunto-operaatioita kutsutaan *aritmeettisiksi* operaatioiksi.

Edellä esitellyillä operaatioilla voidaan siis muuntaa halutun rekisterin arvo nollaksi, kasvattaa halutun rekisterin arvoa yhdellä tai korvata halutun rekisterin arvo toisen rekisterin arvolla. Nämä aritmeettiset operaatiot ovat intuitiivisesti melko helppoja sisäistää. RRK:n neljäs operaatio taas on aavistuksen verran monimutkaisempi.

Halutunlaista ohjelmaa suunnitellessa tulee usein vastaan tilanteita, joissa on hyödyllistä tai jopa välttämätöntä palata aiempaan kohtaan ohjelmassa tai tarvittaessa sivuuttaa tulevia ohjeita. Tällaisia tilanteita varten on olemassa niinsanottu *hyppyoperaatio*.

Määritelmä 4 (**Hyppyoperaatio**). *Jokaiselle* $m, n, q = 1, 2, \ldots$ *on olemassa* hyppyoperaatio $J(m, n, q)$. *Jos ohjelmassa* P *tulee vastaan hyppyoperaatio* $J(m, n, q)$*, niin rekistereitä* R_n *ja* R_m *verrataan keskenään, mutta kaikki rekisterit jätetään entiselleen. Mikäli* $r_m = r_n$*, niin RRK jatkaa* P*:n* q*:nteen*

operaatioon. Jos taas $r_m \neq r_n$, niin RRK siirtyy seuraavaan operaatioon. Jos $r_m = r_n$, mutta hyppy ei ole mahdollnen, koska P:ssä on vähemmän kuin q operaatiota, RRK pysähtyy.

Olkoon RRK:lle annettu ohjelma $P = I_1, I_2, \ldots, I_s$. Tällöin RRK aloittaa annetun ohjelman suorittamisen suorittaen ensin ensimmäisen käskyn I_1. Suoritettuaan käskyn $I_k (k \leq n)$ RRK suorittaa *seuraavan käskyn* joka määritellään seuraavasti:

1. Jos I_k on aritmeettinen operaatio, seuraava käsky on I_{k+1},
2. Jos $I_k = J(m, n, q)$, niin seuraava käsky on I_q, jos $r_m = r_n$ ja I_{k+1} muulloin.

Ohjelma etenee tällä tavoin lopettaen laskemisen vain, jos sillä ei ole enään seuraavaa komentoa. Kun RRK on suorittanut edellisenä käskyn I_k, tämä voi tapahtua seuraavilla tavoilla:

1. Jos $k = s$ (viimeinen komento, jonka P on toimittanut) ja I_s on aritmeettinen komento,
2. Jos $I_k = J(m, n, q)$, $r_m = r_n$ ja $q > s$,
3. Jos $I_k = J(m, n, q)$, $r_m \neq r_n$ ja $k = s$.

Kun laskeminen pysähtyy käskyn I_k jälkeen, RRK:n silloista asetusta $r_1, r_2, r_3, \ldots$ sanotaan *lopputilaksi*.

On tärkeää huomioida, että on olemassa myös ohjelmia, jotka eivät pysähdy ollenkaan. Esimerkiksi ohjelma $S(1)$, $J(1, 1, 1)$ ei pysähdy ikinä, vaan lisää ensimmäiseen rekisteriin yhden ikuisesti.

Nyt on määritelty RRK, sen neljä operaatiota, mitä tarkoitetaan "ohjelmalla" ja koska ohjelma lopettaa toimintansa. Seuraavaksi käymme läpi, kuinka RRK:lla lasketaan ja mitä kaikkea sillä voidaan ylipäätään laskea.

1.1 Rajoittamattomalla rekisterikoneella laskeminen

Jotta RRK:lla voidaan suorittaa lasku-operaatioita, pitää siis olla ohjelma P, jonka *alkuasetuksissa* on rekistereissä $R_1, R_2, R_3, \ldots$ luonnolliset luvut $r_1, r_2, r_3, \ldots$ rekistereiden arvoina. Oletetaan, että ohjelma P sisältää käskyt $I_1, I_2, I_3, \ldots, I_s$, missä $s \in \mathbb{N}$. RRK alkaa laskemisen toimimalla käskyn I_1 mukaan, sitten ohjeen I_2 ja sitten käskyn I_3 ja niin edelleen, kunnes se on käynyt läpi kaikki käskyt, jolloin se pysähtyy, ellei vastaan tule hyppyoperaatiota $J(m, n, q)$. Tällöin RRK toimii kuten $J(m, n, q)$ käskee.

Seuraavaksi käymme askel askeleelta läpi, kuinka RRK käsittelee annettua ohjelmaa annetuissa alkuasetuksissa. Tässä vaiheessa ei ole oleellista, tekeekö ohjelma mitään järkevää, sillä tarkoituksena on ainoastaan tarkastella eri operaatioiden toimintaa käytännössä.

Esimerkki 4. Olkoon ohjelma P seuraavanlainen:

I_1 : $T(3,2)$
I_2 : $S(3)$
I_3 : $T(4,2)$
I_4 : $J(1,4,2)$
I_5 : $Z(1)$

ja olkoon RRK:n alkutila seuraava:

R_1	R_2	R_3	R_4	R_5	R_6	...
4	2	1	3	0	0	...

Tällöin ohjelma suorittaa ensiksi ensimmäisen käskyn, eli se muuntaa rekisterin R_2 arvon 2 rekisterin R_3 arvoksi, eli luvuksi 1. Tämän jälkeen ohjelma suorittaa toisen käskyn, eli se lisää rekisterin R_3 arvoon yhden, jonka jälkeen ohjelma muuntaa rekisterin R_2 arvon 1 rekisterissä R_4 olevaan arvoon, eli lukuun 3. Seuraavaksi ohjelma tarkistaa, onko rekisterien R_1 ja R_4 arvot samat. Koska näin ei ole, ohjelma siirtyy seuraavaan käskyyn, eli se nollaa rekisterin R_1. Koska ohjelmalla ei ole tämän jälkeen seuraavaa käskyä, se pysähtyy.

Ohjelman etenemistä voidaan kuvata seuraavanlaisesti aloittaen alkutilasta, jonka jälkeen ohjelma suorittaa käskyn I_1, sitten käskyn I_2 jne:

R_1	R_2	R_3	R_4	R_5	R_6	...
4	2	1	3	0	0	...

4	1	1	3	0	0	...

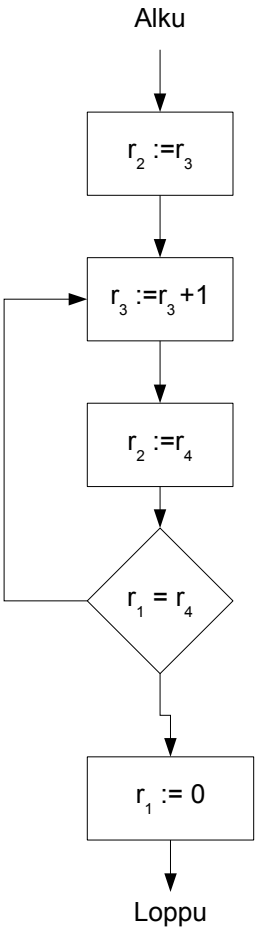
4	1	2	3	0	0	...

4	3	2	3	0	0	...

4	3	2	3	0	0	...

0	1	2	3	0	0	...

Ohjelma P voidaan esittää myös *vuokaaviona* (ks. Kuva 1).



Kuva 1: Rekisterikoneen toiminta

Määritellään muutamia käsitteitä helpottamaan asioiden käsittelyä.

Määritelmä 5. *Olkoon $a_1, a_2, \cdots \in N$ ja olkoon P ohjelma. Tällöin:*

1. *$P(a_1, a_2, \dots)$ tarkoittaa laskentaa ohjelmalla P alkutilalla $a_1, a_2, \dots$;*
2. *$P(a_1, a_2, \dots) \downarrow$ tarkoittaa, että laskenta $P(a_1, a_2, \dots)$ pysähtyy lopulta;*
3. *$P(a_1, a_2, \dots) \uparrow$ tarkoittaa, että laskenta $P(a_1, a_2, \dots)$ ei pysähdy ollenkaan.*

Useimmissa tapauksissa alkutilassa on äärellinen määrä nollasta poikkeavia numeroita ja loput nollia. Tällaisissa tapauksissa merkitään

4. *$P(a_1, a_2, \dots, a_n)$ tarkoittaa samaa kuin $P(a_1, a_2, \dots, a_n, 0, 0, \dots)$.*

Erityisesti

5. *$P(a_1, a_2, \dots, a_n) \downarrow$ tarkoittaa, että $P(a_1, a_2, \dots, a_n, 0, 0, 0, \dots) \downarrow$.*
6. *$P(a_1, a_2, \dots, a_n) \uparrow$ tarkoittaa, että $P(a_1, a_2, \dots, a_n, 0, 0, 0, \dots) \uparrow$.*

Ohjelmaa, joka pysähtyy, kutsutaan *pysähtyväksi* ja ohjelmaa, joka ei pysähdy, kutsutaan *pysähtymättömäksi*.

Olemme nyt käyneet läpi kuinka RRK käytännössä toimii. On kuitenkin vielä täysin auki, miten RRK:lla voidaan tehdä mitään hyödyllistä ja mitä kaikkea sillä voi tehdä.

1.2 RRK-laskettavat funktiot

Olkoon f funktio joukolta $\mathbb{N}^n$ joukolle $\mathbb{N}$ ($n \geq 1$). Haluamme tietää, voiko tämän funktion laskea RRK:lla. Tätä varten tarkastelemme ohjelman $P(a_1, a_2, a_3, \dots, a_n)$ laskutoimituksia. Jos joku näistä toimituksista pysäyttää ohjelman, meillä täytyy olla yksikäsitteinen luku, joka on laskemisen tulos. Varaamme rekisterin R_1 tätä käyttöä varten, joten ohjelman P lopetettua toimintansa r_1 on P:n laskeman laskun tulos.

Koska ohjelma $P(a_1, a_2, a_3, \dots, a_n)$ ei välttämättä pysähdy, sallimme laskettavuuden määritelmän koskevan myös "funktiota" $f\colon \mathbb{N}^n \to \mathbb{N}$, joiden määrittelyjoukko ei välttämättä ole koko $\mathbb{N}^n$. Tällä tarkoitetaan esimerkiksi osittain määriteltyjä funktioita. Tulemme määrittelemään relevantin laskettavuuden siten, että laskeminen päättyy vain ja ainoastaan f:n määrittelyjoukkoon kuuluvilla arvoilla.

Määrittelemme laskettavuuden seuraavasti:

Määritelmä 6. *Olkoon f osittain määritelty funktio $\mathbb{N}^n$:ltä $\mathbb{N}$:lle.*

1. *Olkoon P ohjelma ja olkoon $a_1, a_2, \ldots, a_n, b \in N$.*

   (a) *Ohjelma $P(a_1, a_2, \ldots, a_n)$* pysähtyy arvoon *b, jos $P(a_1, a_2, \ldots, a_n) \downarrow$ ja jos ohjelman loppuarvo b on R_1:ssä. Tällaisesta tilanteesta käytämme merkintää $P(a_1, a_2, \ldots, a_n) \downarrow b$.*

   (b) *Ohjelma P RRK-laskee funktion f, jos jokaiselle $a_1, a_2, \ldots, a_n, b$ $P(a_1, a_2, \ldots, a_n) \downarrow b$ täsmälleen silloin kun $(a_1, a_2, \ldots, a_n) \in Dom(f)$ ja $f(a_1, a_2, \ldots, a_n) = b$. Siis erityisesti $P(a_1, a_2, \ldots, a_n) \downarrow$ täsmälleen silloin kun $(a_1, a_2, \ldots, a_n) \in Dom(f)$.*

2. *Funktio f on* RRK-laskettava, *jos on olemassa ohjelma, joka RRK-laskee f:n.*

Merkitään RRK-laskettavien funktioiden joukkoa $\mathscr{U}$:lla ja n-paikkaisten laskettavien funktioiden joukkoa $\mathscr{U}_n$:llä. Tästä eteenpäin *laskettavalla* funktiolla tarkoitetaan nimenomaan RRK-laskettavaa funktiota (ellei toisin mainita).

Katsomme nyt kuinka RRK:lla lasketaan käytännössä. Toteutamme ohjelman, joka laskee summan $x + y$.

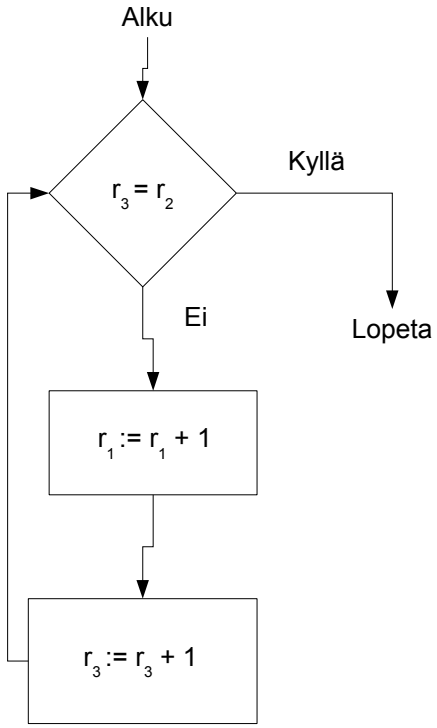
Esimerkki 5. Laskemme summan $x + y$ lisäämällä x:ään y kertaa luvun 1. Ohjelman alkutila on tällöin muotoa:

R_1	R_2	R_3	R_4	R_5	R_6	$\ldots$
x	y	0	0	0	0	$\ldots$

Ohjelma P, joka suorittaa tämän operaation voidaan tehdä esimerkiksi seuraavalla tavalla (ks. Kuva 2):

I_1 : $J(3, 2, 5)$
I_2 : $S(1)$
I_3 : $S(3)$
I_4 : $J(1, 1, 1)$

Ohjelma tarkistaa ensiksi, onko $y = r_3 = 0$. Mikäli näin on, ohjelma lopettaa toimintansa ja lasku $x+0 = x$ on selvästi oikein. Jos näin ei ole, niin ohjelma lisää lukuihin r_1 ja r_3 luvun 1, jonka jälkeen ohjelma palaa alkuun. Nyt ohjelma tarkistaa uudelleen onko $y = r_3$, jonka jälkeen ohjelma pysähtyy mikäli näin on, mutta jatkaa toimintaansa mikäli näin ei ole. Ohjelma siis käy yksitellen läpi luvut $0, 1, 2, \ldots$ kunnes se löytää luvun, joka on yhtä suuri y:n



Kuva 2: x+y

kanssa. Koska $y \in \mathbb{N}$, tällainen luku on välttämättä olemassa, jolloin ohjelma pysähtyy välttämättä jossakin vaiheessa.

Näin ollen $x + y$ on laskettava funktio.

1.2.1 Ratkeavat predikaatit ja ongelmat

Matematiikassa tavallinen tehtävä on *päätellä* täyttävätkö tietyt numerot jotkin ehdot. Voidaan esimerkiksi tarkastella, onko x luvun y:n monikerta. Tähän tehtävään voidaan määrittää algoritmi, joka antaa tulokseksi 1 mikäli näin on, ja 0, mikäli näin ei ole. Tällaista funktiota voidaan merkitään seuraavalla tavalla:

$$f(x,y) = \begin{cases} 1, & \text{jos } x \text{ on } y\text{:n moninkerta,} \\ 0, & \text{jos } x \text{ ei ole } y\text{:n moninkerta.} \end{cases} \tag{1}$$

Voimme sanoa, että predikaatti "x on y:n monikerta" on *algoritmisesti ratkeava* jos kyseinen funktio f on laskettava.

Määritelmä 7. *Yleisemmin voimme määritellä, että $M(x_1, x_2, \ldots, x_n)$ on luonnollisten lukujen n-paikkainen predikaatti. Tällöin* karakteristinen funktio $c_M(\mathrm{x})$, *missä* $\mathrm{x} = (x_1, x_2, \ldots, x_n)$, *määritellään seuraavasti:*

$$c_M(\mathrm{x}) = \begin{cases} 1, & \textit{jos } M(\mathrm{x}) \textit{ on tosi,} \\ 0, & \textit{jos } M(\mathrm{x}) \textit{ ei ole tosi.} \end{cases} \tag{2}$$

Esimerkki 6. Predikaatti $x \neq 2$ on laskettava, sillä on olemassa ohjelma P, joka antaa tulokseksi 1 mikäli predikaatti pitää paikkaansa ja 0 mikäli se ei pidä paikkaansa. Predikaatin karakteristinen funktio on

$$g(x) = \begin{cases} 1, & \text{jos } x \neq 2, \\ 0, & \text{jos } x = 2 \end{cases} \tag{3}$$

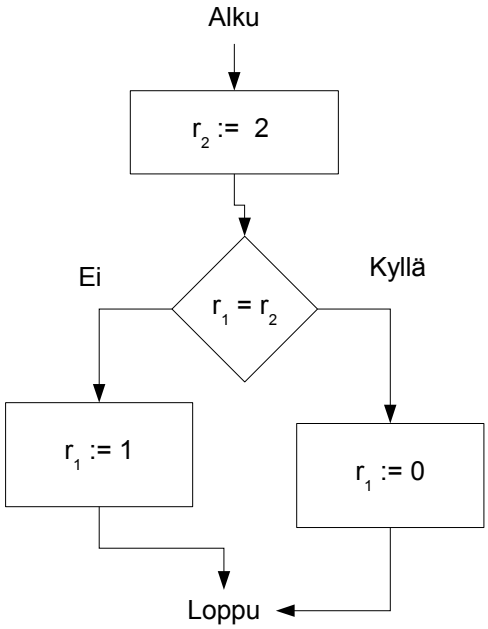
ja ohjelma P, joka laskee funktion $g(x)$ voidaan esittää esimerkiksi seuraavasti (ks. Kuva 3):

I_1 : $S(2)$
I_2 : $S(2)$
I_3 : $J(1, 2, 7)$
I_4 : $Z(1)$
I_5 : $S(1)$
I_6 : $J(1, 1, 8)$
I_7 : $Z(1)$

Tässä P lisää ensiksi tyhjään rekisteriin luvun 2, jonka jälkeen se tarkistaa, onko annettu luku x sama kuin luku 2. Jos näin on, se tulostaa luvun 0 ja jos näin ei ole, se tulostaa luvun 1. Tämä on selvästi haluttu ohjelma.

1.2.2 Laskettavuus muissa määrittelyjoukoissa

Koska RRK käsittelee vain luonnollisia lukuja, meidän laskettavuuden ja todistuvuuden määritelmämme pätevät vain luonnollisia lukuja käsitteleviin funktioihin. Jotta voisimme käsitellä myös muiden määrittelyjoukkojen funktioita, on käytettävä erityistä *koodausta*. Määrittelyjoukon D koodaus on yksikäsitteinen injektio $\alpha\colon D \to \mathbb{N}$. Tällöin objekti $d \in D$ on *koodattu*



Kuva 3: $x \neq 2$

luonnollisella luvulla $\alpha(D)$. Olkoon $f\colon D \to D$ ja f^* funktio, joka kuvaa objektin $d \in Dom(f)$ koodin $f(d)$:n koodiksi; tällöin f^* koodaa f:n $\mathbb{N}$:ltä $\mathbb{N}$:lle. Nimenomaisesti:

$$f^* = \alpha \circ f \circ \alpha^{-1}. \tag{4}$$

Nyt voimme laajentaa RRK-määritelmää käsittämään joukon D sanomalla, että f on *laskettava*, mikäli f^* on luonnollisten lukujen joukossa määritelty laskettava funktio.

Esimerkki 7. Voimme määritellä kokonaislukujen joukolle $\mathbb{Z}$ seuraavanlaisen koodauksen funktion α avulla:

$$\alpha = \begin{cases} 2n, & \text{jos n} \geq 0, \\ -2n-1, & \text{jos n} < 0. \end{cases} \tag{5}$$

Tällöin α^{-1} on

$$\alpha^{-1} = \begin{cases} \frac{1}{2}m, & \text{jos } m \text{ on parillinen,} \\ -\frac{1}{2}(m+1), & \text{jos } m \text{ on pariton.} \end{cases} \tag{6}$$

Tarkastelkaamme nyt funktiota $f(x) = 2x$. Tällöin $f^*(\alpha(x)) = \alpha(f(x))$. Jos $x \geq 0$, niin $f^*(\alpha(x)) = f^*(2x)\alpha(f(x)) = \alpha(2x) = 4x$. Jos taas $x < 0$, niin $f^*(\alpha(x)) = f^*(-2x-1)\alpha(f(x)) = \alpha(2x) = -4x - 1$. Täten siis

$$f^*(x) = \begin{cases} 2x, & \text{jos } x \text{ on parillinen,} \\ 2x-1, & \text{jos } x \text{ on pariton.} \end{cases} \tag{7}$$

1.3 Ohjelmien yhdistäminen

Oletetaan, että meillä on ohjelmat P ja O, ja että haluamme ohjelman, joka tekee ensin sen, mitä ohjelma P tekee ja sen jälkeen sen, mitä ohjelma O tekee. Ensimmäinen ajatus halutunlaisen ohjelman luomiseksi olisi vain liittää ohjelma O ohjelman P perään. Tämä ei kuitenkaan välttämättä riitä halutun ohjelman luomiseksi, sillä on kaksi asiaa, jotka täytyy ottaa huomioon.

Olkoon $P = I_1, I_2, I_3, \ldots, I_s$. Tällöin P pysähtyy, kun ohjelma olisi siirtymässä johonkin komentoon I_v, missä $v > s$. Koska ohjelma O on sijoitettu heti P:n perään, alkaa ohjelma toimimaan oikein vain, jos $v = s + 1$. Näin ollen meidän on varmistettava, että P pysähtyy vain komentoon I_{s+1}. Tällaista ohjelmaa kutsutaan *standardimuotoiseksi* ohjelmaksi. On selvää, että muunlainen lopetus ohjelmalle on mahdollista ainoastaan hyppyoperaation avulla.

Määritelmä 8. *Olkoon ohjelma $P = I_1, I_2, I_3, \ldots, I_s$. Tällöin P on* standardimuodossa*, jos jokaiselle P:n hyppyoperaatiolle $J(m, n, q)$ on voimassa $q \leq s + 1$.*

Lemma 1. *Jokaiselle ohjelmalle P on olemassa standardimuodossa oleva ohjelma P^* siten, että jokainen ohjelman P^* laskutoimitus on identtinen vastaavaan laskutoimituksen kanssa P:ssä mahdollisesti ohjelman pysähtymisen tapaa lukuunottamatta. Kaikille $a_1, \ldots, a_n, b$ pätee*

$$P(a_1, a_2, a_3, \ldots, a_n) \downarrow b \text{ jos ja vain jos } P^*(a_1, a_2, a_3, \ldots, a_n) \downarrow b$$

ja täten $f_P^{(n)} = f_{P^*}^{(n)}\ \forall n > 0$, missä merkinnällä $f_P^{(n)}$ tarkoitetaan n-paikkaista funktiota, jonka ohjelma P laskee.

Todistus. Olkoon $P = I_1, I_2, I_3, \ldots, I_s$. Jotta saataisiin P^* ohjelmasta P muunnetaan vain hyppyoperaatiota siten, että jos hyppääminen pysäyttää

ohjelman, niin hyppy tapahtuu käskyyn I_{s+1}. Joten olkoon $P^* = I_1^*, I_2^*, \ldots, I_s^*$ sellainen ohjelma, että jos I_k ei ole hyppyoperaatio, niin $I_k^* = I_k$, ja jos $I_k = J(m,n,q)$, niin

$$I_k^* = \begin{cases} I_k, & \text{jos } q \leq s+1, \\ J(m,n,s+1), & \text{jos } q > s+1. \end{cases} \tag{8}$$

Täten P^* on selvästi vaaditunlainen.

□

Oletetaan nyt, että ohjelmat P ja O ovat standardimuodossa. Tällöin ongelmaksi muodostuvat O:n hyppyoperaatiot, sillä kun normaalisti operaatio $J(m,n,q)$ käskee hyppäämään q:nteen operaatioon (mikäli $r_m = r_n$), niin ohjelmien yhdistämisen jälkeen q:s operaatio ei enään ole sama operaatio kun aiemmin. Tämän vuoksi ohjelmia yhdistettäessä on O:n jokainen hyppyoperaatio vaihdettava muodosta $J(m,n,q)$ muotoon $J(m,n,q+s)$, jolloin hyppyoperaatiot viittaavat alunperin suunniteltuihin operaatioihin.

Näin ollen voimme muodostaa standardimuodossa olevien ohjelmien *yhdistelmiä*.

Määritelmä 9. *Olkoon P ja Q standardimuodossa olevat ohjelmat, joiden pituudet ovat s ja t. Näiden ohjelmien* yhdistelmä *(kirjoitetaan PQ) on ohjelma $I_1, I_2, I_3, \ldots, I_s, I_{s+1}, \ldots, I_{s+t}$, missä $P = I_1, I_2, I_3, \ldots, I_s$ ja Q:n operaatiot $I_{s+1}, \ldots, I_{s+t}$ ovat siten muutettuja, että jokainen hyppyoperaatio $J(m,n,q)$ on korvattu $J(m,n,s+q)$:lla.*

Nyt meillä on siis käytössämme työkalut, joiden avulla voimme yhdistää erilaisia ohjelmia saadaksemme aikaan monimutkaisempia ohjelmia ilman, että meidän tarvitsee välttämättä kirjoittaa yhtä pitkää ohjelmaa alusta asti. On kuitenkin huomioitava sellainen asia, että kirjoittaessamme ohjelmaa PQ, voi tulla vastaan tilanne, jossa Q kirjoittaa muistiin erilaisia arvoja ja tällöin on huolehdittava siitä, että Q:n tarvitsemat "varastorekisterit" eivät ole olleet P:n käytössä. Tästä asiasta voidaan huolehtia seuraavalla tavalla:

Koska P on äärellinen, on olemassa pienin sellainen luku u, että mitään rekistereistä R_v, missä $v > u$, ei mainita ohjelmassa P. Jos siis $Z(n)$, $S(n)$, $T(m,n)$ ja $J(m,n,q)$ ovat P:n komentoja, niin $m,n \leq u$. Tällöin kaikissa P:n vaiheissa R_v:n sisältö pysyy muuttumattomana, kun $v > u$. Tällöin kirjoittaessamme Q:ta, voimme varastoida tietoa rekistereihin R_v, missä $v > u$, ilman, että se vaikuttaa P:n toimintaan. Tällaista lukua u merkitsemme $\rho(P)$.

Tämän lisäksi määrittelemme hieman merkintätapoja, jotka helpottavat asioiden käsittelyä vastaisuudessa. Oletetaan, että P on standardimuodossa

oleva ohjelma, joka laskee funktion $f(x_1, x_2 \ldots, x_n)$. Jos P on jonkin suuremman ohjelman aliohjelma, voi olla hyödyllisempää varastoida funktion $f(x_1, x_2 \ldots, x_n)$ syötteet $x_1, x_2 \ldots, x_n$ ohjelman normaalisti ehdottamien rekistereiden $R_1, R_2, \ldots, R_n$ sijaan rekistereihin $R_{l_1}, R_{l_2}, \ldots, R_{l_n}$. Tämä voi olla tarpeen, sillä funktion $f(x_1, x_2, \ldots, x_n)$ tulostetta voidaan tarvita myöhemmin rekisterissä R_l ennemmin kuin rekisterissä R_1. Voi olla myös niin, että ohjelman P työrekisterit $R_1, R_2 \ldots, R_{\rho_{(P)}}$ voivat sisältää informaatiota, joka sekoittaa laskennan. Siksi on hyödyllistä muokata P:tä niin, että se osaa ottaa huomioon edellämainitun kaltaiset tilanteet.

Kirjoitamme $P[l_1, l_2 \ldots, l_n \rightarrow l]$ tarkoittamaan ohjelmaa, joka laskee funktion $f(R_{l_1}, R_{l_2} \ldots, R_{l_n})$, mutta joka tulostaa vastauksen rekisteriin R_l. Ohjelman toiminta on kuvattu kuvassa 4. Näin ollen ainoat rekisterit, joihin ohjelma P voi teoriassa vaikuttaa ovat rekisterit $R_1, R_2, \ldots, R_{\rho_{(P)}}$ ja R_l. On hyvä huomioida, että määritellessämme ohjelman $P[l_1, l_2, \ldots, l_n \rightarrow l]$ oletamme, että rekisterit $R_{l_1}, R_{l_2} \ldots, R_{l_n}$ eroavat rekistereistä $R_1, R_2, \ldots, R_n$. Mikäli näin ei automaattisesti jostain syystä ole, tilanne on helppo palauttaa halutunlaiseksi.

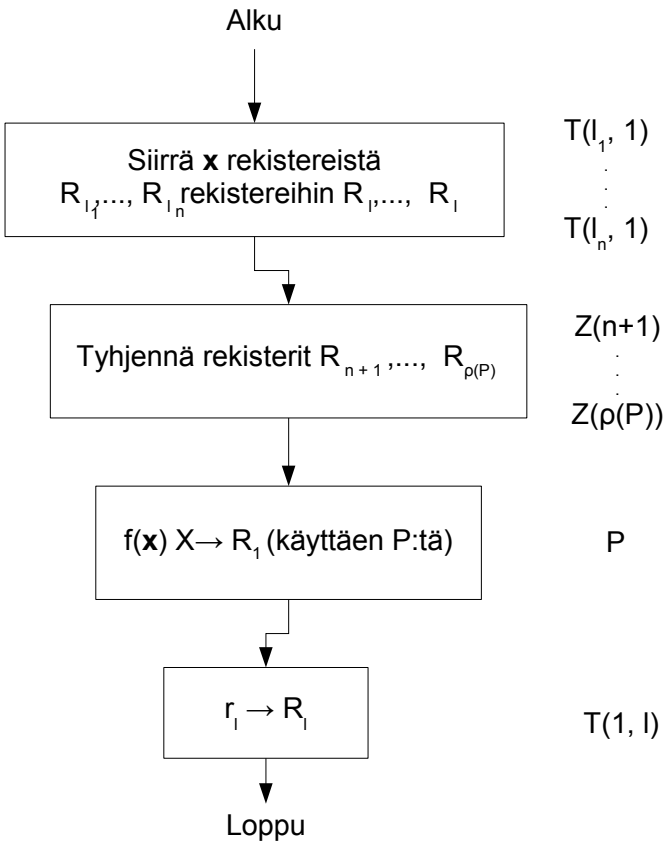
2 Osittainrekursiiviset funktiot

Edellä olemme käsitelleet rajoittamatonta rekisterikonetta ja kuinka sillä voidaan laskea. Seuraavaksi tutustumme toisenlaiseen tapaan lähestyä laskettavuuden käsitettä.

Rekursio on tapa määritellä funktioita siten, että funktion arvo määräytyy sen edellisistä arvoista. Intuitiivisesti helppo esimerkki rekursion käsitteestä on Fibonaccin lukujono. Tässä lukujonon ensimmäinen arvo on 0, toinen arvo on 1 ja lukujonon seuraavat arvot ovat kahden edellisen arvon summa. Tällöin funktion ensimmäisiksi arvoiksi tulevat $0, 1, 1, 2, 3, 5, 8, 13, \ldots$ Tämä lukujono voidaan määritellä rekursiivisena funktiona seuraavalla tavalla:

$$F(n) = \begin{cases} 0, & \text{kun n=0,} \\ 1, & \text{kun n=1,} \\ F(n-1) + F(n-2), & \text{kun n>1.} \end{cases} \tag{9}$$

Toisaalta sama lukusarja voidaan myös esittää funktiona



Kuva 4: $P[l_1, \ldots, l_n \to l]$

$$F(n) = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}. \tag{10}$$

Nämä kaksi funktiota siis kuvaavat samaa lukujonoa, mutta on helppo havaita, että rekursion avulla määritelty funktio on selvästi yksinkertaisempi sekä merkitä, että ymmärtää intuitiivisesti.

Tässä luvussa määrittelemme täsmällisesti osittainrekursiivisten funktioiden joukon. Osittainrekursiivisten funktioiden määrittelyyn tarvitsemme edellä esitellyn, mutta täsmällisesti määrittelemättömän, rekursion käsitteen lisäksi muutaman muun käsitteen. Käymme tarvittavat käsitteet ensiksi läpi

ja sen jälkeen todistamme, että RRK-laskettavien funktioiden joukko $\mathscr{U}$ on *suljettu* näiden operaatioiden suhteen. Tällöin olemme todistaneet, että kaikki osittainrekursiiviset funktiot voidaan esittää RRK:n avulla.

2.1 Perusfunktiot

Osittainrekursiivisten funktioiden joukkoa varten meidän on ensiksi määriteltävä ns. *perusfunktiot*, joita yhdistelemällä voidaan määritellä muita funktioita.

Määritelmä 10. *Perusfunktiot ovat seuraavat kolme funktiota:*

1. *nollafunktio* $\boldsymbol{0}$ ($\boldsymbol{0}(x) = 0$ *kaikille* x)*;*
2. *seuraajafunktio* $x + 1$*;*
3. *projektiofunktio* $U_i^n(x_1, x_2, x_3, \ldots, x_n) = x_i$ *kaikille* $n \geq 1$ *ja* $1 \leq i \leq n$.

Näistä kolmesta funktiosta ensimmäinen siis antaa aina arvoksi 0, toinen kasvattaa haluttua lukua yhdellä ja viimeinen antaa useampipaikkaisen funktion arvoksi halutun paikan arvon.

Seuraavaksi käymme läpi *korvaamisen*, *rekursion* ja *minimalisoinnin*.

2.2 Korvaaminen

Uusia laskettavia funktioita voidaan tehdä jo tunnetuista laskettavista funktioista esimerkiksi järjestämällä uudelleen tai identifioimalla sen muuttujia tai lisäämällä ylimääräisiä muuttujia.

Määritelmä 11 (Korvaaminen). *olkoon* $f(y_1, \ldots, y_k)$ *ja* $g_1(\boldsymbol{x}), \ldots, g_k(\boldsymbol{x})$ *RRK-laskettavia funktioita, missä* $\boldsymbol{x} = (x_1, \ldots, x_n)$*. Tällöin on olemassa funktio* $h(\boldsymbol{x}) \simeq f(g_1(\boldsymbol{x}), \ldots, g_k(\boldsymbol{x}))$.

2.3 Rekursio

Rekursio on tapa, jolla funktion arvo lasketaan sen edellisistä arvoista ja mahdollisesti muista jo tunnetuista funktioista.

Määritelmä 12 (Rekursio). *Olkoon* $\boldsymbol{x} = (x_1, \ldots, x_n)$ *ja olkoon* $f(\boldsymbol{x})$ *ja* $g(\boldsymbol{x}, y, z)$ *funktioita. Tällöin on olemassa funktioista* $f(\boldsymbol{x})$ *ja* $g(\boldsymbol{x}, y, z)$ *kostruoitava, yksikäsitteinen funktio* $h(\boldsymbol{x}, y)$ *joka toteuttaa rekursio-ehdot:*

1. $h(\boldsymbol{x}, 0) \simeq f(\boldsymbol{x})$
2. $h(\boldsymbol{x}, y + 1) \simeq g(\boldsymbol{x}, y, h(\boldsymbol{x}, y))$.

Funktion haluttu arvo saadaan laskemalla annetusta alku-arvosta seuraava arvo, jonka jälkeen lasketaan sitä seuraava arvo ja tätä jatketaan kunnes saadaan funktion haluttu arvo.

On huomioitava, että mikäli f ja g eivät ole täydellisiä funktioita, ei h ole myöskään välttämättä täydellinen funktio. Tällöin h:n määrittelyjoukolla on seuraavat ominaisuudet:

1. $(\mathbf{x}, 0) \in Dom(h)$ joss $\mathbf{x} \in Dom(f)$,
2. $(\mathbf{x}, y+1) \in Dom(h)$ joss $(\mathbf{x}, y) \in Dom(f)$ ja $(\mathbf{x}, y, h(\mathbf{x}, y)) \in Dom(g)$.

Lisäksi kannattaa huomata, että rekursion määritelmässä ei ole kyse kehäpäättelystä, sillä laskettaessa funktion $h(\mathbf{x}, y)$ arvoa käytetään aina arvoja, jotka tunnetaan jo.

Esimerkki 8. Esimerkiksi lukujen x ja y yhteenlasku saadaan ilmoitettua seuraavalla tavalla: kaikilla x, y pätee $x + 0 = x$ ja $x + (y + 1) = (x + y) + 1$. Täten yhteenlasku, eli funktio $h(x, y) = x + y$, määritellään rekursiivisesti funktioilla $f(x) = x$ ja $g(x, y, z) = z + 1$.

Voimme määritelmässä 10 määritellyistä perusfunktioista ja korvaamisesta ja rekursiosta määritellä *primitiivirekursiiviset funktiot* siten, että primitiivirekursiivisten funktioiden joukko $\mathscr{PR}$ on pienin joukko funktioita, joka on suljettu perusfunktioiden, korvaamisen ja rekursion suhteen. Määritelläksemme osittainrekursiiviset funktiot, meidän on määriteltävä vielä minimalisointi.

2.4 Minimalisointi

Oletetaan, että meillä on funktio $f(\mathbf{x}, y)$ ja haluamme tietää pienimmän sellaisen y:n arvon, jolla $f(\mathbf{x}, y) = 0$. Tätä varten määrittelemme funktion $G(\mathbf{x})$ = pienin sellainen y, että $f(\mathbf{x}, y) = 0$ Tässä voi tulla esille seuraavat kaksi ongelmaa:

1. Voi olla, että millään y ei käy niin, että $f(\mathbf{x}, y) = 0$.
2. Oletetaan, että f on laskettava. Käytetään seuraavanlaista luonnollista algoritmia laskemaan $G(\mathbf{x})$: "Lasketaan $f(\mathbf{x}, 0)$, $f(\mathbf{x}, 1)$, ... kunnes löydetään sellainen y, että $f(\mathbf{x}, y) = 0$". Tämä proseduuri ei kuitenkaan välttämättä pysähdy jos f ei ole täydellinen, vaikka tällainen y olisi olemassa, sillä voi esimerkiksi olla, että $f(\mathbf{x}, 0)$ ei ole määritelty, mutta $f(\mathbf{x}, 1) = 0$.

Määrittelemme siis *minimalisaatio-operaattorin* μ seuraavalla tavalla:

Määritelmä 13 (**Minimalisointi**). *Jokaiselle funktiolle* $f(\boldsymbol{x}, y)$ *on olemassa funktio*

$$\mu y(f(\boldsymbol{x}, y) = 0) = \begin{cases} \text{pienin sellainen } y, \text{ että } f(\mathbf{x}, z) \text{ on määritelty kaikilla} \\ z \leq y \text{ ja } f(\mathbf{x}, y) = 0, \text{ jos tällainen } y \text{ on olemassa;} \\ \text{määrittelemätön muulloin.} \end{cases} \tag{11}$$

Edellä määriteltyjen operaatioiden avulla määritellään *osittanrekursiiviset funktiot*.

Määritelmä 14. Osittainrekursiivisten funktioiden *joukko* $\mathscr{R}$ *on pienin joukko funktioita, joka sisältää perusfunktiot* $\boldsymbol{0}$, $x + 1$ *ja* U_i^n *ja on suljettu korvaamisen, rekursion ja minimalisoinnin suhteen.*

Seuraavaksi tarkastelemme niisanottua *Ackermannin funktiota* [3, s. 362] esimerkkinä funktiosta, joka on osittainrekursiivinen, mutta ei primitiivirekursiivinen.

Esimerkki 9. Ackermannin funktio

Määritellään funktio $\psi(x, y)$ rekursion avulla seuraavalla tavalla:

$$\psi(x, y) = \begin{cases} y + 1, & \text{kun x=0,} \\ \psi(x - 1, 1), & \text{kun y=0,} \\ \psi(x - 1, \psi(x, y - 1)), & \text{muulloin.} \end{cases} \tag{12}$$

Voimme havainnollistaa funktion käyttäytymistä merkitsemällä $f_x(y) = \psi(x, y)$, jolloin saadaan funktiojono $f_0, f_1, f_2, \ldots$, missä

$$\begin{aligned} f_0(y) &= y + 1 \\ f_1(y) &= y + 2 \\ f_2(y) &= 2y + 3 \\ f_3(y) &= 2^y - 3 \\ f_4(y) &= \underbrace{y^{\cdot^{\cdot^{\cdot^{y}}}}}_{y+3} - 3 \end{aligned}$$

Intuitiivisesti tämä funktio on melko helppo ymmärtää laskettavaksi, mutta sen määrittelyn sisältämä eräänlainen tuplarekursio tekee sen mahdottomaksi määritellä ilman minimalisointia. Todistus siitä, että funktio on osittainrekursiivinen, mutta ei primitiivirekursiivinen sivuutetaan.

3 Osittainrekursiivisten funktioiden laskettavuus

Tämän kappaleen tavoitteena on osoittaa, että jokainen osittainrekursiivinen funktio on RRK-laskettava. Todistaaksemme, että $\mathscr{R} \subseteq \mathscr{U}$, meidän täytyy osoittaa, että perusfunktiot 0, $x+1$ ja U_i^n ovat laskettavia funktioita ja että $\mathscr{U}$ on suljettu korvaamisen, rekursion ja minimalisoinnin suhteen.

3.1 Perusfunktiot

Lause 1. *Perusfunktiot* $\boldsymbol{0}$*,* $x+1$ *ja* U_i^n *ovat RRK-laskettavia funktioita.*

Todistus. Jokaisen funktion laskettavaksi osoittamiseksi riittää esittää ohjelma, joka laskee k.o. funktion.

1. Funktion $\boldsymbol{0}$ laskee ohjelma $Z(1)$.
2. Funktion $x+1$ laskee ohjelma $S(1)$.
3. Funktion $U_i^n(x_1, x_2, \ldots, x_n)$ laskee ohjelma $T(i,1)$.

□

Seuraavaksi osoitamme, että $\mathscr{U}$ on suljettu sekä korvaamisen, rekursion, että minimalisoinnin suhteen.

3.2 Korvaaminen

Lause 2. *Olkoon* $f(y_1, \ldots, y_k)$ *ja* $g_1(\boldsymbol{x}), \ldots, g_k(\boldsymbol{x})$ *RRK-laskettavia funktioita, missä* $\boldsymbol{x} = (x_1, \ldots, x_n)$*. Tällöin funktio* $h(\boldsymbol{x}) \simeq f(g_1(\boldsymbol{x}), \ldots, g_k(\boldsymbol{x}))$ *on RRK-laskettava.*

Todistus. Olkoon $F, G_1, \ldots, G_k$ normaalimuodossa olevia ohjelmia, jotka laskevat funktiot $f, g_1, \ldots, g_k$. Väitteen todistamiseksi riittää muodostaa ohjelma H, joka laskee h:n. Haluttu ohjelma toimii siten, että kun $\mathbf{x}$ on määritelty, käytetään ohjelmia $G_1, \ldots, G_k$ laskemaan funktiot $g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})$ ja sitten ohjelmaa F laskemaan $f(g_1(\mathbf{x}), \ldots, g_k(\mathbf{x}))$.

Merkitään $m = \max(n, k, \rho(F), \rho(G_1), \ldots, \rho(G_k))$, missä $\rho(G)$ tarkoittaa pienintä sellaista lukua u, että mitään rekistereistä R_v ei tarvita, missä $v > u$. Tällä tavalla varmistetaan, että tarvitsemamme varastorekisterit ovat tyhjiä, eikä hyödyllistä informaatiota mene hukkaan. Talletetaan $\mathbf{x}$ rekistereihin $R_{m+1}, \ldots, R_{m+n}$ ja rekistereitä $R_{m+n+1}, \ldots, R_{m+n+k}$ käytetään tallentamaan arvot $g_i(x)$, kun $i = 1, 2, \ldots, k$. Nämä rekisterit unohdetaan

laskettaessa ohjelmilla $F, G_1, \ldots, G_k$. Ohjelman konfiguraatio näyttää tällöin seuraavalta:

$R_1 \ldots R_m$	$R_{m+1} \ldots R_{m+n}$	R_{m+n+1}	R_{m+n+2}	$\ldots$	R_{m+n+k}	
$\ldots$	$\mathbf{x}$	$g_1(\mathbf{x})$	$g_2(\mathbf{x})$	$\ldots$	$g_k(\mathbf{x})$	$\ldots$

Ohjelma H voidaan ilmaista seuraavasti:

$$
\begin{array}{l}
T(1, m+1) \\
\vdots \\
T(n, m+n) \\
G_1[m+1, m+2, \ldots, m+n \to m+n+1] \\
\vdots \\
G_k[m+1, m+2, \ldots, m+n \to m+n+k] \\
F[m+n+1, \ldots, m+n+k \to 1]
\end{array}
$$

Ohjelma H siis laskee ensin funktiot $g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})$ varastoiden jokaisen funktion vastauksen varastorekistereihin $R_{m+n+1}, \ldots, R_{m+n+k}$. Tämän jälkeen ohjelmaa laskee funktion $f(y_1, \ldots, y_k)$ siten, että se ottaa funktion muuttujien arvoiksi funktioiden $g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})$ arvot niistä varastorekistereistä, joihin ne on tallennettu. Ohjelman toiminta on kuvattu kuvassa 5.

Selvästi $H(x)$ pysähtyy täsmälleen silloin, kun kaikki laskut $G_i(\mathbf{x})$ $(1 \leq i \leq k)$ ja $F(g_1(\mathbf{x}), \ldots, g_k(\mathbf{x}))$ pysähtyvät, mikä on vaadittu tulos.

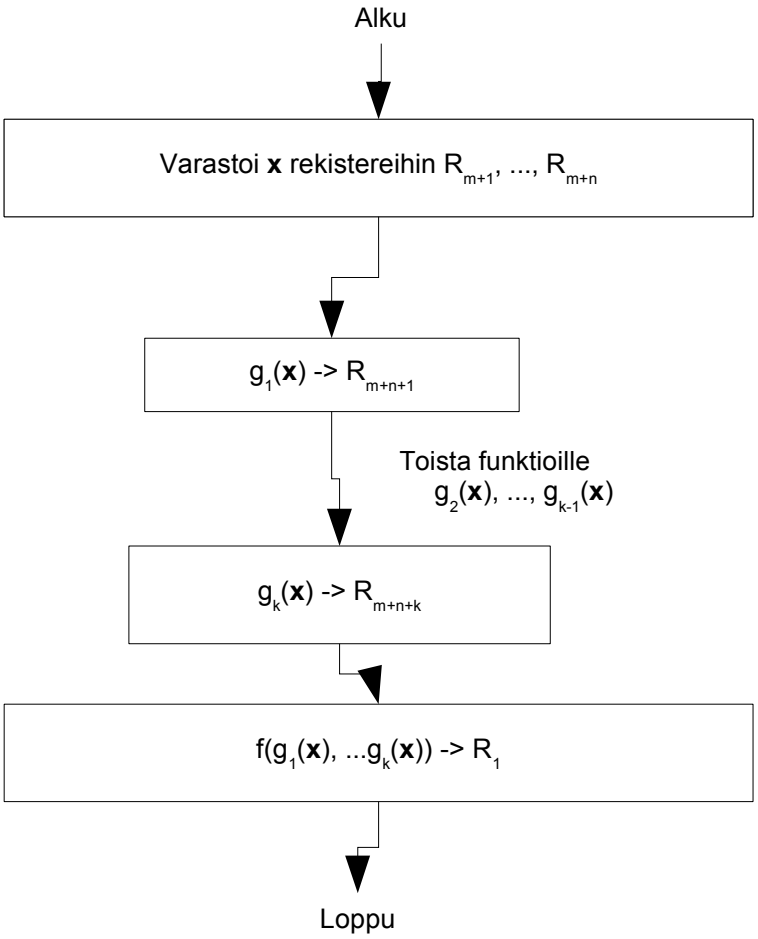
□

Huom. Edellisessä lauseessa on huomioitava, että $h(\mathbf{x})$ on määritelty täsmälleen silloin, kun $g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})$ ovat kaikki määriteltyjä ja $(g_1(\mathbf{x}), \ldots, g_k(\mathbf{x}))$ $\in Dom(f)$; täten jos f ja $g_1, \ldots, g_k$ ovat kaikki täydellisiä funktioita, niin h on täydellinen funktio.

Esimerkki 10. Funktiosta $f(y_1, y_2)$ voidaan muokata esimerkiksi funktiot:

$h_1(x_1, x_2) \simeq f(x_2, x_1)$	(uudelleenjärjestäminen),
$h_2(x) \simeq f(x, x)$	(identifiointi),
$h_3(x_1, x_2, x_3) \simeq f(x_2, x_3)$	(ylimääräisen muuttujan lisääminen).

Seuraava lause, joka on lauseen 2 sovellus, osoittaa, että esimerkiksi esimerkin 10 operaatioiden avulla laskettavista funktioista saadut uudet funktiot ovat laskettavia.



Kuva 5: Korvaaminen

Lause 3. *Olkoon $f(y_1, \ldots, y_k)$ laskettava funktio ja muuttujien $x_1, \ldots, x_n$ k-jono (jossa on mahdollisesti toistoja) on $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$. Tällöin h:n määräämä funktio*

$$h(x_1, \ldots, x_n) \simeq f(x_{i_1}, x_{i_2}, \ldots, x_{i_k})$$

on laskettava.

Todistus. Merkitsemällä $\mathbf{x} = (x_1, \ldots, x_n)$ saamme

$$h(\mathbf{x}) \simeq f(U^n_{i_1}(\mathbf{x}), U^n_{i_2}(\mathbf{x}) \ldots, U^n_{i_k}(\mathbf{x})).$$

Tämä taas on laskettava määritelmän 10 ja lauseen 2 nojalla.

□

Tämän lauseen avulla näemme, että lause 2 pätee kun f:ään sijoitetut funktiot $g_1, \ldots, g_k$ eivät ole välttämättä kaikkien muuttujien $x_1, \ldots, x_n$ funktioita.

3.3 Rekursio

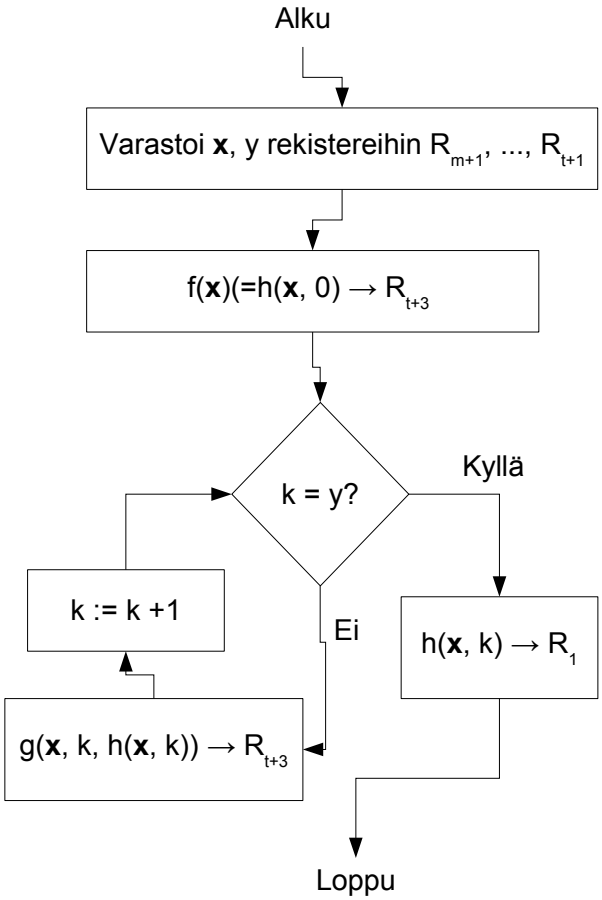
Lause 4. *Olkoon $f(\boldsymbol{x})$ ja $g(\boldsymbol{x}, y, z)$ laskettavia funktioita , missä $\boldsymbol{x} = (x_1, \ldots, x_n)$. Tällöin funktio $h(\boldsymbol{x}, y)$, joka saadaan f:stä ja g:stä rekursion avulla, on laskettava.*

Todistus. Olkoon F ja G normaalimuodossa olevia ohjelmia, jotka laskevat funktiot $f(\mathbf{x})$ ja $g(\mathbf{x}, y, z)$. Teemme ohjelman H funktiolle $h(\mathbf{x}, y)$ käyttämällä rekursion määritelmää. Kun alkuperäiset asetukset ovat $x_1, \ldots, x_n, y, 0, 0, \ldots,$ H laskee ensin $h(\mathbf{x}, 0)$:n käyttämällä F:ää. Sitten, jos $y \neq 0$, H käyttää G:tä laskeakseen peräkkäin $h(\mathbf{x}, 1)$:n, $h(\mathbf{x}, 2)$:n, ...ja $h(\mathbf{x}, y)$:n ja pysähtyy tämän jälkeen.
Olkoon $m = \max(n + 2, \rho(F), \rho(G))$. Aloitamme varastoimalla $\mathbf{x}$:n ja y:n arvot rekistereihin $R_{m+1}, \ldots, R_{m+n+1}$; seuraavaa kahta rekisteriä käytetään varastoimaan numeroiden k ja $h(\mathbf{x}, k)$ $(k = 0, 1, 2, \ldots, y)$ nykyiset arvot. Merkitsemällä $t = m + n$ proseduurin konfiguraatio näyttää seuraavalta:

$R_1 \ldots R_m$	$R_{m+1} \ldots R_t$	R_{t+1}	R_{t+2}	R_{t+3}	$\ldots$
$\ldots$	$\mathbf{x}$	y	k	$h(x,k)$	$\ldots$

kun $k = 0$ aluksi.

Ohjelman toiminta on kuvattu kuvassa 6.
Tämä proseduuri voidaan kääntää ohjelmaksi H, joka laskee h:n:



Kuva 6: Rekursio

$$
\begin{array}{ll}
 & T(1, m+1) \\
 & \vdots \\
 & T(n+1, m+n+1) \\
 & F[1, 2, \ldots, n \to t+3] \\
I_q: & J(t+2, t+1, p) \\
 & G[m+1, \ldots, m+n, t+2, t+3 \to t+3] \\
 & S(t+2) \\
 & J(1, 1, q) \\
I_p: & T(t+3, 1)
\end{array}
$$

Täten h on laskettava.

□

3.4 Minimalisointi

Lause 5. *Olkoon funktio $f(\boldsymbol{x}, y)$ laskettava. Tällöin funktio $g(\boldsymbol{x}) = \mu y(f(\boldsymbol{x}, y) = 0)$ on myös laskettava.*

Todistus. Olkoon $\mathbf{x} = (x_1, \ldots, x_n)$ ja F standardimuodossa oleva ohjelma, joka laskee funktion $f(\mathbf{x}, y)$. Olkoon $m = \max(n + 1, \rho(F))$. Kirjoitamme ohjelman G, joka ilmaisee luonnollisen algoritmin g:lle: kun $k = 0, 1, 2, \ldots$ lasketaan $f(\mathbf{x}, k)$:ta kunnes löydetään sellainen k, että $f(\mathbf{x}, k) = 0$. Tämä k on haluttu ulostulo.

Arvot $x_1, x_2, \ldots, x_n$ sekä k:n nykyinen arvo tallennetaan rekistereihin $R_{m+1}, \ldots, R_{m+n+1}$ ennenkuin lasketaan $f(\mathbf{x}, k)$, jonka jälkeen konfiguraatio näyttää seuraavanlaiselta ($R_{m+1} \ldots R_{m+n+2}$ ovat varastorekistereitä):

$R_1 \ldots R_m$	$R_{m+1} \ldots R_{m+n}$	R_{m+n+1}	R_{m+n+2}	$\ldots$
$\ldots$	$\mathbf{x}$	k	0	$\ldots$

Tällöin k on alunperin nolla ja r_{m+n+2} on aina nolla.
Ohjelman toiminta on kuvattu kuvassa 7.
Ohjelma G, joka toteuttaa g:n näyttää seuraavalta:

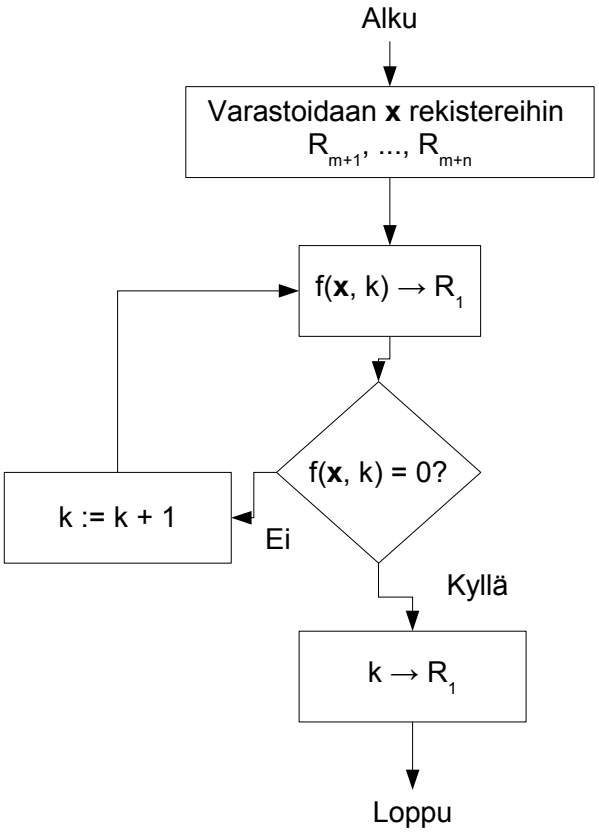
$$
\begin{array}{ll}
 & T(1, m+1) \\
 & \vdots \\
 & T(n, m+n) \\
I_p: & F[m+1, m+2, \ldots, m+n+1 \to 1] \\
 & J(1, m+n+2, q) \\
 & S(m+n+1) \\
 & J(1, 1, p) \\
I_q: & T(m+n+1, 1)
\end{array}
$$

□

Lause 6. $\mathscr{R} \subseteq \mathscr{U}$

Todistus. Väite seuraa lauseista 1, 2, 4 ja 5

□



Kuva 7: Minimalisointi

4 Turingin kone

Kolmas, ja viimeinen tässä työssä käsiteltävä, tapa määritellä laskettavuuden käsite täsmällisesti on niin sanottu *Turingin kone*. Se on kenties tunnetuin laskettavuuden täsmällisistä määritelmistä. Koneen suunnitteli Englantilainen matemaatikko Alan Turing vuonna 1936.

4.1 Turingin koneen määrittely

Turingin koneen rakenne muistuttaa hieman RRK:ta, sillä myös siinä varastoidaan merkkejä samankaltaiselle nauhalle kun RRK:ssa. Nauhan lisäksi Turingin koneessa on lukupää, joka on aina jossakin tilassa, ja joka lukee kohdallaan olevan symbolin ja toimii sen jälkeen sille annettujen ohjeiden

mukaisesti.

Määritelmä 15. *Turingin kone M rakentuu seuraavista osista:*

1. *Äärellinen aakkosto $\Sigma = \{s_0, s_1, s_2, \ldots, s_n\}$. Varaamme symbolin s_0 merkitsemään tyhjää ruutua. Nauhalla merkitsemme sitä symbolilla ∇. Lisäksi varaamme symbolin s_1 niinsanotuksi "merkkisymboliksi" (tästä eteenpäin vain "merkki") ja merkitsemme sitä symbolilla* 1.

2. *Äärellinen määrä tiloja $Q = \{q_1, q_2, q_3, \ldots, q_m\}$.*

3. *Äärellinen määrä käskyjä $I_1, I_2, \ldots, I_k$, joista jokainen on seuraavaa muotoa:*

   (a) *$q_a s_j s_k q_l$: Luettuaan symbolin s_j tilassa q_a oleva kone pyyhkii luetun symbolin ja kirjoittaa tilalle symbolin s_k, jonka jälkeen kone siirtyy tilaan q_l.*

   (b) *$q_a s_j R q_l$: Luettuaan symbolin s_j tilassa q_a oleva kone siirtyy yhden ruudun oikealle, jonka jälkeen se siirtyy tilaan q_l.*

   (c) *$q_a s_j L q_l$: Luettuaan symbolin s_j tilassa q_a oleva kone siirtyy yhden ruudun vasemmalle, jonka jälkeen se siirtyy tilaan q_l.*

Tulee huomoida, että ohjelman käskyjen on oltava yksikäsitteiset, eli jokainen pari $q_a s_j$ viittaa korkeintaan yhteen nelikkoon $q_a s_j \alpha \beta$, sillä muuten kone ei tiedä minkä ohjeen se suorittaisi.

Kone toimii siten, että sille annetaan nauha ja se asetetaan tiettyyn alkutilaan. Tämän jälkeen kone lukee lukupäällä lukupään kohdalla olevan symbolin, jonka jälkeen se toimii kyseiselle symbolille määrätyn käskyn mukaisesti, jatkaen tällä tavoin kunnes sillä ei ole enään käskyä, jonka se voisi toteuttaa, jolloin ohjelma pysähtyy. On täysin mahdollista, että näin ei käy koskaan ja tällöin ohjelma jatkaa toimintaansa loputtomiin.

Tarkastellaksemme Turingin koneen toimintaa käytännössä teemme ohjelman, joka muuntaa annetun nauhan kaikki symbolit nolliksi, kunnes se törmää tyhjään ruutuun, jonka jälkeen ohjelma pysähtyy.

Esimerkki 11. Olkoon M Turingin kone, jonka aakkosto on $\{\nabla, 0, 1\}$ ja sen ainoa mahdollinen tila on q_1. Tällöin ohjelma, joka muuntaa nauhan kaikki symbolit nolliksi, on muotoa:

$q_1 1 0 q_1$
$q_1 0 R q_1$

Oletetaan, että M:lle on annettu seuraavanlainen nauha (symbolilla $\downarrow$ tarkoitetaan lukupään sijaintia):

		$\downarrow$					
$\ldots$	∇	1	1	0	1	∇	$\ldots$

Alussa kone on tilassa q_1. Koneen käynnistyessä se lukee nauhalta symbolin 1, jonka jälkeen se tarkistaa ohjeistaan, mitä sen pitää tehdä luettuaan symbolin 1. Kone havaitsee, että sen tulee korvata symboli 1 symbolilla 0 ja pysyä tilassa q_1. Tämän jälkeen tilanne nauhalla on seuraava:

		$\downarrow$					
$\ldots$	∇	0	1	0	1	∇	$\ldots$

Nyt kone lukee nauhalta symbolin 0, jonka jälkeen se katsoo ohjeistaan, mitä sen tulee tehdä seuraavaksi. Ohjeiden mukaisesti kone siirtyy yhden ruudun oikealle ja pysyy tilassa q_1. Tämän jälkeen ohjelma lukee seuraavan symbolin ja koska se on 1, ohjelma korvaa sen symbolilla 0 ja siirtyy askeleen oikealle. Tämän jälkeen tilanne näyttää seuraavalta:

				$\downarrow$			
$\ldots$	∇	0	0	0	1	∇	$\ldots$

Seuraavaksi kone lukee nauhalta symbolin 0 ja siirtyy suoraan seuraavaan ruutuun. Tämän jälkeen kone lukee symbolin 1, korvaa sen symbolilla 0 ja siirtyy oikealle. Koska seuraava ruutu on tyhjä, kone ei tiedä mitä sen pitäisi tehdä, joten se pysähtyy ja koneen lopputila näyttää seuraavalta:

						$\downarrow$	
$\ldots$	∇	0	0	0	0	∇	$\ldots$

4.2 Turingin koneen toiminta

Vaikka Turingin koneen aakkosto voi periaatteessa olla minkälainen hyvänsä, kunhan se on äärellinen, käytetään tavallisesti yksinkertaisuuden vuoksi aakkostoa, jossa on vain symbolit ∇ ja 1, sillä tämän aakkoston avulla voidaan ilmaista täsmälleen samat asiat kun laajemmissakin aakkostoissa.

Kun koneen merkistö koostuu vain symboleista ∇ ja 1, tarvitaan jokin

tapa esittämään kokonaislukuja. Käytämme tapaa, jossa $x+1$ peräkkäistä merkkiä (eli symbolia 1) tarkoittaa lukua x. Yksi ylimääräinen merkki tarvitaan, jotta voidaan erottaa luku 0 tyhjästä ruudusta merkitsemällä sitä yhdellä merkillä.

Laskeaksemme n-paikkaisen funktion $f(x_1, x_2, \ldots, x_n)$ tarvitsemme tavan eroittaa luvut $x_1, x_2, \ldots, x_n$ toisistaan nauhalla. Tämän teemme erottamalla kaksi lukua toisistaan tyhjällä ruudulla.

Määrittelemme tässä työssä Turingin koneen alkutilan sellaiseksi, että funktion $f(x_1, x_2, \ldots, x_n)$ syötteet ovat nauhalla järjestyksessä $x_1, x_2, \ldots, x_n$ vasemmalta lukien ja lukupää on viimeistä muuttujan x_n varaamasta ruudusta katsottuna seuraava ruutu oikealle päin. Tämä määritelmä on täysin mielivaltainen, joten toiminnaltaan erilaisia Turingin koneita on olemassa lukuisia.

On huomioitava myös sellainen asia, että periaatteessa emme voi tietää, että onko nauhalla syötteen lisäksi muita aakkoston symboleja, joten oletamme, että alkutilassa nauha on tyhjä syötettä lukuunottamatta.

Turing-laskettavuuden määrittelemme seuraavalla tavalla.

Määritelmä 16. *Funktio $f(x_1, x_2, \ldots, x_n)$ on Turing-laskettava, mikäli on olemassa Turingin kone M siten, että mikäli koneeseen on annettu syötteet $x_1, x_2, \ldots, x_n$ (jolloin lukupää on siis x_n:n oikealla puolella), pysähtyy kone seuraamalla M:n ohjeita tilaan, jossa tuloste $f(x_1, x_2, \ldots, x_n)$ on lukupään vasemmalla puolella.*

Esimerkki 12. Funktio $f(x, y) = x + y$ on Turing-laskettava, sillä on olemassa ohjelma, joka laskee tämän laskun. Ohjelma voi olla esimerkiksi seuraavanlainen:

$q_1 \nabla L q_2$
$q_2 1 \nabla q_3$
$q_3 \nabla L q_4$
$q_4 1 L q_4$
$q_4 \nabla L q_5$
$q_5 1 L q_5$
$q_5 \nabla R q_6$
$q_6 1 \nabla q_7$
$q_7 \nabla R q_8$
$q_8 1 R q_8$
$q_8 \nabla 1 q_9$
$q_9 1 R q_9$

Nauhalle on siis syötetty luku x ja luku y, jolloin nauhalla on siis yhteensä $x + y + 2$ ykköstä ja yksi tyhjä ruutu ykkösien välissä. Ohjelma toimii siten,

että lukupää liikkuu yhden ruudun vasemmalle, jonka jälkeen lukupää pyykii kohtaamansa ykkösen. Tämän jälkeen lukupää liikkuu vasemmalle, kunnes törmää tyhjään ruutuun ja jatkaa siitä vieläkin vasemmalle, kunnes kohtaa seuraavan tyhjän ruudun. Seuraavaksi lukupää siirtyy yhden ruudun oikealle, pyyhkii siinä olevan ykkösen, siirtyy oikealle kunnes kohtaa tyhjän ruudun ja korvaa sen ykkösellä. Lopuksi lukupää siirtyy oikealle, kunnes kohtaa tyhjän ruudun ja pysähtyy siihen. Lopputuloksena lukupään vasemmalla puolella on $x + y + 1$ ykköstä, joka on haluttu tulos. Näinollen funktio $x + y$ on Turing-laskettava.

4.3 Osittainrekursiivisten funktioiden Turing-laskettavuus

Olemme nyt tarkastelleet Turingin koneen toimintaa ja seuraavaksi todistamme, että kaikki osittainrekursiiviset funktiot ovat Turing-laskettavia. Todistuksen luonne on vastaavan kaltainen kun osoittaessamme osittainrekursiivisten funktioiden olevan myös RRK-laskettavia. Koska todistus olisi yksityiskohtaisesti läpi käytynä liian pitkä tätä työtä varten, emme käy todistusta läpi yksityiskohtaisesti. Tarkka todistus löytyy esimerkiksi Martin Davisin kirjasta Computability and unsolvability [1958].

Todistaaksemme, että jokainen osittainrekursiivinen funktio on Turing-laskettava osoitamme, että perusfunktiot ovat Turing-laskettavia, ja että Turing-laskettavien funktioiden joukko on suljettu korvaamisen, rekursion ja minimalisoinnin suhteen.

Lause 7. *Perusfunktiot* 0, $x + 1$ *ja* U_i^n *ovat Turing-laskettavia.*

Todistus. Osoitamme, että kukin näistä funktioista on Turing-laskettava.

1. Nollafunktio

   Oletetaan, että nauhalle on syötetty luku x ja haluamme, että kone pyyhkii x:n ja tulostaa sen sijaan luvun 0. Tämä onnistuu esimerkiksi seuraavalla ohjelmalla:

   $q_0 \nabla L q_1$
   $q_1 1 L q_2$
   $q_2 1 R q_3$
   $q_3 1 \nabla q_0$
   $q_2 \nabla R q_4$
   $q_4 1 R q_5$

   Ohjelma toimii siten, että aluksi lukupää siirtyy yhden ruudun vasemmalle, jossa se lukee ykkösen. Tämän jälkeen lukupää liikkuu toisen

ruudun vasemmalle tarkistamaan, onko siellä ykkönen vai ei. Jos kyseinen ruutu on ykkönen, lukupää palaa tyhjentämään edellisen ruudun ja aloittaa toimintansa alusta. Jos taas lukupää ei törmää ykköseen liikuttuaan ensin kaksi ruutua vasemmalle, lukupää palaa takaisin kahden ruudun verran ja pysähtyy. Näin ohjelma siis tyhjentää ruutuja, kunnes se törmää ykköseen, jonka vasemmalla puolella ei ole toista ykköstä. Tämän tilanteen tullessa vastaan, lukupään liikuttua ensin kaksi ruutua oikealle, on lukupään vasemmalla puolella yksi ykkönen, joka on haluttu tulos.

2. Projektiofunktio

   Projektiofunktio $U_i^n(x_1, x_2, x_3, \ldots, x_n) = x_i$ saadaan aikaiseksi siten, että tehdään ohjelma, joka siirtää lukupään luvun x_n vierestä luvun x_{n-1} viereen, ja suoritetaan tämä ohjelma $n - i$ kertaa. Sopiva ohjelma voidaan tehdä esimerkiksi seuraavalla tavalla:

   $q_0 \nabla L q_1$
   $q_1 1 L q_2$

   Tässä ohjelmassa lukupää siis siirtyy ensiksi yhden ruudun vasemmalle ja sen jälkeen vasemmalle aina lukiessaan ykkösen. Kun lukupää lukee seuraavan kerran tyhjän ruudun, ohjelma pysähtyy, joka on haluttu tulos.

3. Seuraajafunktio

   Seuraajafunktio saadaan lisäämällä nauhalle yksi ykkönen ja siirtämällä lukupäätä yhden askeleen oikealle tämän jälkeen:

   $q_0 \nabla 1 q_1$
   $q_1 1 R q_2$

□

Seuraavaksi osoitamme, että Turing-laskettavien funktioiden joukko on suljettu korvaamisen, rekursion ja minimalisoinnin suhteen.

Lause 8. *Olkoon $f(y_1, \ldots, y_k)$ ja $g_1(\boldsymbol{x}), \ldots, g_k(\boldsymbol{x})$ RRK-laskettavia funktioita, missä $\boldsymbol{x} = (x_1, \ldots, x_n)$. Tällöin funktio $h(\boldsymbol{x}) \simeq f(g_1(\boldsymbol{x}), \ldots, g_k(\boldsymbol{x}))$ on Turing-laskettava.*

Todistus. Olkoon

$$h(\mathbf{x}) = f(g_1(\mathbf{x}), \ldots, g_k(\mathbf{x}))$$

ja oletetaan, että funktiot $g_1(\mathbf{x}), \ldots, g_k(\mathbf{x}), f$ ovat laskettavia koneilla $M_1, \ldots, M_m, M_{m+1}$. Tällöin h voidaan laskea ohjelmalla, jonka toiminta on kuvattu vuokaaviossa 8. Käytännössä ohjelma toimii siten, että lasketaan syötteiden oikealle puolelle ensin $g_1(\mathbf{x})$, sitten $g_2(\mathbf{x})$ jne siten, että tulosteet tallennetaan riviin laskemisjärjestyksessä, kunnes on laskettu kaikki funktiot $g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})$. Tämän jälkeen lasketaan saatuja arvoja $g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_k(\mathbf{x})$ käyttämällä haluttu funktio, eli $f(g_1(\mathbf{x}), \ldots, g_k(\mathbf{x}))$, ja lopuksi poistetaan arvot $g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_k(\mathbf{x})$, sekä alkuperäinen syöte ja siirretään lukupää saadun arvon oikealle puolelle. Täten funktio $h(\mathbf{x})$ on Turing-laskettava.

□

Lause 9. *Olkoon $f(\boldsymbol{x})$ ja $g(\boldsymbol{x}, y, z)$ laskettavia funktioita , missä $\boldsymbol{x} = (x_1, \ldots, x_n)$. Tällöin funktio $h(\boldsymbol{x}, y)$, joka saadaan f:stä ja g:stä rekursion avulla, on Turing-laskettava.*

Todistus. Tarkastellaan rekursiivisesti määriteltyjä funktioita

$$h(\mathbf{x}, 0) \simeq f(\mathbf{x}),$$
$$h(\mathbf{x}, y+1) \simeq g(\mathbf{x}, y, h(\mathbf{x}, y)).$$

Oletetaan, että f ja g ovat laskettavia funktioita, jotka laskevat koneet M_1 ja M_2. Laskemme funktion h vuokaavion 9 mukaisesti. Ideana on laskea h:n peräkkäisiä arvoja yksi kerrallaan sen edellistä arvoa käyttäen, kunnes olemme saavuttaneet halutun arvon. Pysyäksemme mukana iteraatiokierroksista, käytämme laskuria s, joka on aluksi arvossa y ja jota piennennetään yhdellä jokaisella kierroksella, sekä laskuria t, joka on aluksi 0, mutta jota kasvatetaan jokaisen askeleen jälkeen yhdellä kunnes $t = y$. Täten siis kokoajan $s + t = y$. Kun $t = y$, ohjelma poistaa nauhalta kaiken laskennan tulosta lukuunottamatta, jonka jälkeen lukupää siirtyy vastauksen oikealle puolelle. Täten nauhamme näyttää seuraavalta:

$$x_1 \nabla \ldots \nabla x_n \nabla y \nabla s \nabla t \nabla f(x_1, \ldots, x_n, t).$$

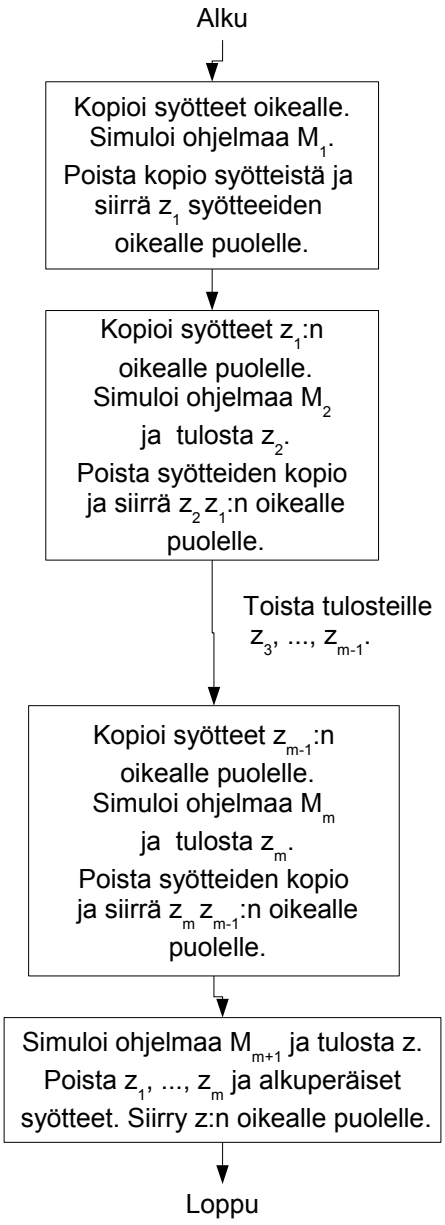
Näinollen funktio $h(\mathbf{x}, y)$ on Turing-laskettava.

□

Lause 10. *Olkoon funktio $f(\boldsymbol{x}, y)$ laskettava. Tällöin funktio $g(\boldsymbol{x}) = \mu y(f(\boldsymbol{x}, y) = 0)$ on myös laskettava.*

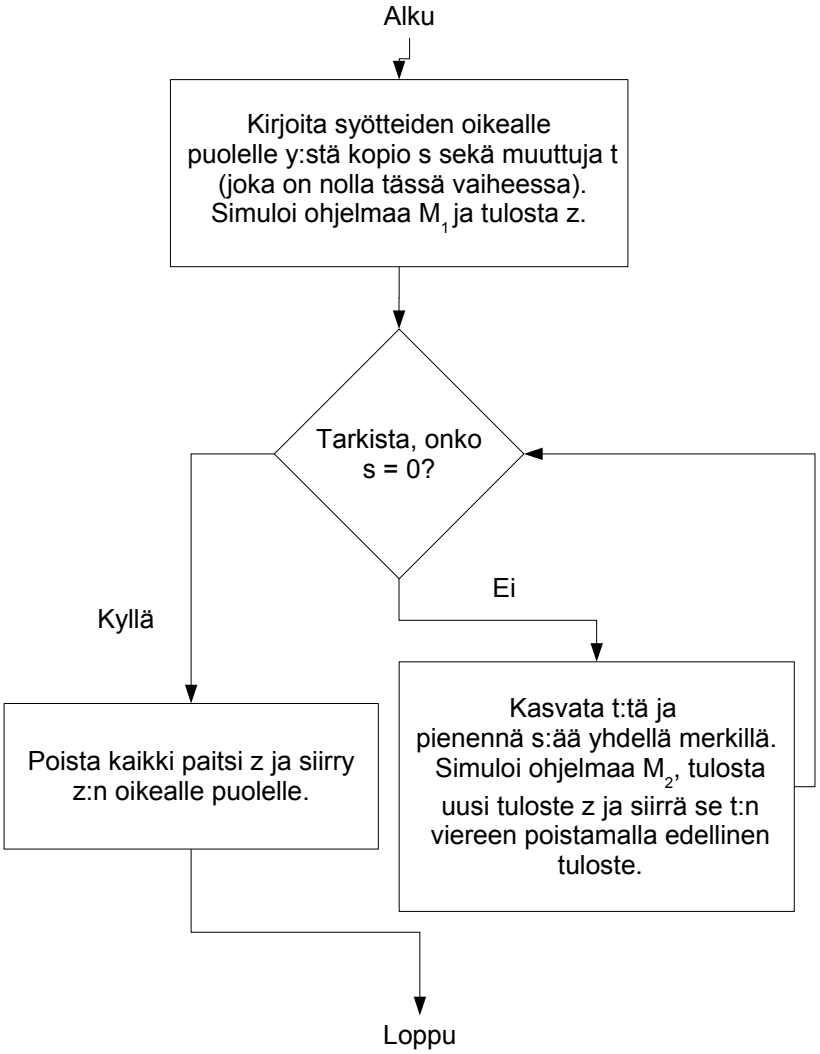
Todistus. Olkoon

$$g(\mathbf{x}) = \mu y(f(\mathbf{x}, y) = 0)$$



Kuva 8: Korvaaminen

ja oletetaan, että f on laskettava funktio ja sen laskee kone M. Tällöin funktion g laskee kone, jonka toiminta on kuvattu vuokaaviossa 10. Koneen idana on laskea funktion $f(\mathbf{x}, y)$ peräkkäisiä arvoja alkaen arvosta $y = 0$, kunnes saavutetaan sellainen y, että $f(\mathbf{x}, y) = 0$. Tämä y on etsitty arvo. Jos taas



Kuva 9: Rekursio

tällaista arvoa ei ole, niin kone jatkaa toimintaansa loputtomiin. Näinollen funktio $g(\mathbf{x}) = \mu y(f(\mathbf{x}, y) = 0)$ on Turing-laskettava.

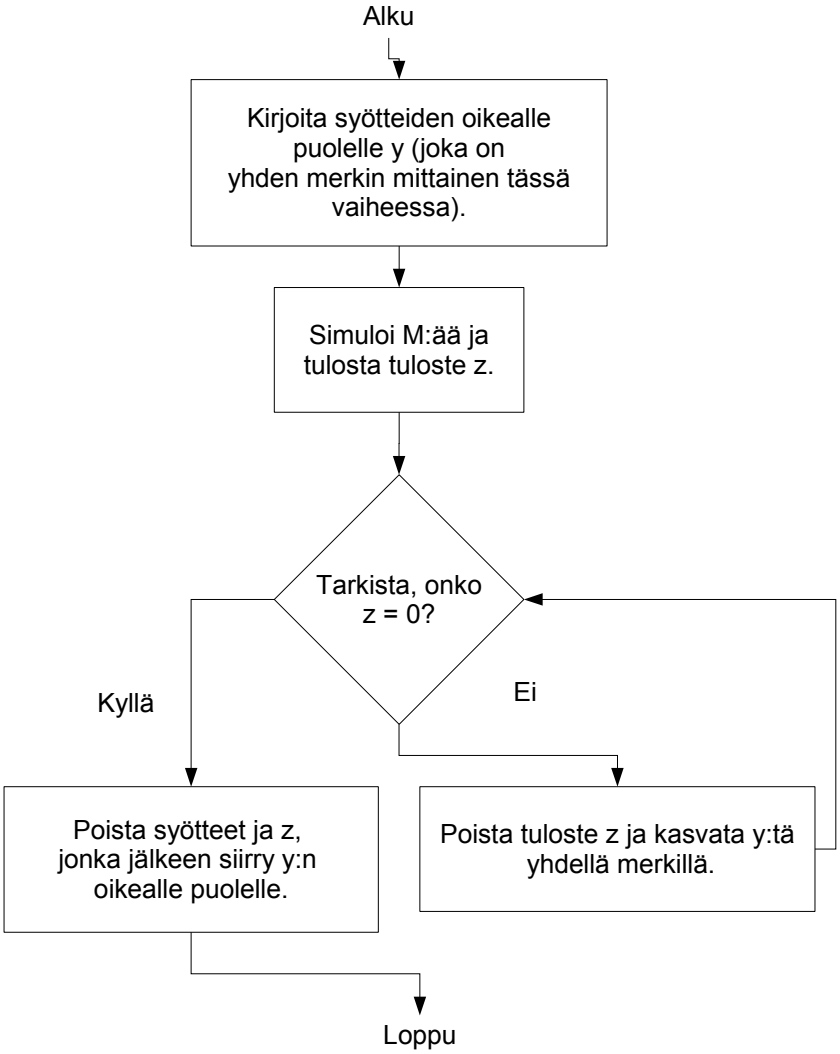
□

Edellisistä lauseista voimme johtopäätöksenä esittää, että $\mathscr{R} \subseteq \mathscr{T}$.

Lause 11. $\mathscr{R} \subseteq \mathscr{T}$.

Todistus. Väite seuraa lauseista 7, 8, 9 ja 10.

□



Kuva 10: Minimalisointi

5 Laskettavien funktioiden joukkojen yhtäpitävyys

Tämän kappaleen tarkoituksena on osoittaa, että $\mathscr{T} = \mathscr{U} = \mathscr{R}$. Tähän mennessä olemme jo todistaneet, että $\mathscr{R} \subseteq \mathscr{U}$ ja että $\mathscr{R} \subseteq \mathscr{T}$. Tämän lisäksi meidän on vielä todistettava väitteet toiseen suuntaan, eli että $\mathscr{U} \subseteq \mathscr{R}$ ja että $\mathscr{T} \subseteq \mathscr{R}$.

Lause 12. $\mathscr{U} \subseteq \mathscr{R}$.

Todistus. Todistaaksemme, että $\mathscr{U} \subseteq \mathscr{R}$ oletetaan, että $f(x)$ on RRK-laskettava funktio, jonka laskee ohjelma $P = I_1, I_2, \ldots, I_s$. Olkoon seuraavat funktiot yhteydessä ohjelmaan P.

$$c(\mathbf{x}, t) = \begin{cases} R_1\text{:n sisältö } t\text{:n askeleen jälkeen} \\ \text{laskettaessa } P(\mathbf{x})\text{:ää, jos } P(\mathbf{x}) \text{ ei ole vielä pysähtynyt;} \\ \\ R_1\text{:n loppusisältö, jos } P(\mathbf{x}) \text{ on pysähtynyt alle } t\text{:ssä askeleessa.} \end{cases} \tag{13}$$

$$j(\mathbf{x}, t) = \begin{cases} \text{seuraavan askeleen järjestysluku, kun ohjelmaa } P(\mathbf{x}) \text{ on käyty } t \\ \text{askelta, jos } P(\mathbf{x}) \text{ ei ole pysähtynyt korkeinaan } t\text{:ssä askeleessa;} \\ \\ 0, \text{ jos } P(\mathbf{x}) \text{ on pysähtynyt korkeintaan } t\text{:ssä askeleessa.} \end{cases} \tag{14}$$

Selvästi c ja j ovat täydellisiä funktioita.

Jos $f(\mathbf{x})$ on määritelty, niin $P(\mathbf{x})$ pysähtyy täsmälleen t_0:n askeen jälkeen, missä

$$t_0 = \mu t(j(\mathbf{x}, t) = 0)$$

ja tällöin

$$f(\mathbf{x}) = c(\mathbf{x}, t_0).$$

Toisaalta, jos $f(\mathbf{x})$:ää ei ole määritelty, niin $P(\mathbf{x})$ ei pysähdy ja niin $j(\mathbf{x}, t)$ ei ole koskaan nolla. Tällöin $\mu t(j(\mathbf{x}, t) = 0)$:ää ei ole määritelty. Täten kummassakin tapauksessa saamme

$$f(\mathbf{x}) \simeq c(\mathbf{x}, \mu t(j(\mathbf{x}, t) = 0)).$$

Osoittaaksemme, että f on osittainrekursiivinen, riittää osoittaa, että c ja j ovat rekursiivisia funktioita. Selvästi nämä funktiot ovat laskettavia informatiivisessa mielessä, sillä voimme seurata laskentaa $P(\mathbf{x})$ t-askelta. On kuitenkin melko työlästä, vaikkei kovin vaikeaa, osoittaa yksityiskohtaisesti, että j ja c ovat rekursiivisia. Tämä tarkempi todistus löytyy Cutlandin kirjasta [1, s. 51]. Täten f on osittainrekursiivinen.

□

Lause 13. $\mathscr{R} = \mathscr{U}$.

Todistus. Väite seuraa lauseista 6 ja 12.

□

Seuraavaksi osoitamme, että $\mathscr{T} \subseteq \mathscr{R}$. Tätä varten määrittelemme apulauseen, jonka tarkan todistuksen sivuutamme tilan puutteen vuoksi.

Lause 14. *On olemassa sellainen primitiivirekursiivinen funktio U ja sellaiset primitiivirekursiiviset predikaatit T_n ($n \geq 1$), että jokaiselle rekursiiviselle n-muuttujaiselle funktiolle f on olemassa yksikäsitteinen luonnollinen luku e (jota kutsutaan f:n* indeksiksi*), jolle pätee:*

1. $\forall x_1, \ldots, \forall x_n \exists y T_n(e, x_1, \ldots, x_n, y)$
2. $f(x_1, \ldots, x_n) = U(\mu y T_n(e, x_1, \ldots, x_n, y))$.

Todistus. Todistuksen ideana on yhdistää yksikäsitteisesti luvut funktioihin ja laskutoimituksiin siten, että predikaatti $T_n(e, x_1, \ldots, x_n, y)$ ilmaisee väitteen: y on funktion, jolla on indeksi e, arvon laskemisen numero syötteillä $x_1, x_2, \ldots, x_n$. Tämän avulla funktio $\mu y T_n(e, x_1, \ldots, x_n, y)$ antaa yhden tällaisen laskennan luvun ja funktio U erottaa tulosteen arvon siitä. Tämä asia on kuitenkin liian pitkä todistaa tätä työtä varten. Yksityskohtaista todistusta varten katso [2, s. 90-96].

□

Lause 15. $\mathscr{T} \subseteq \mathscr{R}$

Todistus. Lauseen 14 perusteella voimme määritellä primitiivirekursiivisen funktion $T_n(e, x_1, \ldots, x_n, y)$ siten, että y koodaa lukuun e viittaavan Turingin koneen syötteillä $x_1, \ldots, x_n$ suorittaman laskun. Olkoon U sellainen primitiivirekursiivinen funktio, että jos y koodaa laskennan, niin $U(y)$ on arvo, joka kirjoitetaan lukupään vasemmalle puolelle y:n koodaaman laskennan viimeisessä konfiguraatiossa. Jos f on funktio, joka on koodattu luvulla e, niin silloin f on rekursiivinen, koska

$$f(x_1, \ldots, x_n) = U(\mu y T_n(e, x_1, \ldots, x_n, y)).$$

□

Lause 16. $\mathscr{T} = \mathscr{R}$

Todistus. Väite seuraa lauseista 11 ja 15.

□

Nyt voimme tehdä sen johtopäätöksen, että tämän työn tärkein väite, eli että RRK-laskettavien funktioiden, osittainrekursiivisten funktioiden ja Turing-laskettavien funktioiden joukot olisivat yhtäpitäviä, on tosi.

Lause 17. $\mathscr{T} = \mathscr{U} = \mathscr{R}$

Todistus. Väite seuraa lauseista 13 ja 16.

□

6 Churchin teesi

Olemme edellä määritelleen "laskettavuuden" kolmella toisistaan poikkeavalla tavalla. Nämä kaikki kolme järjestelmää toimivat hyvinkin erilaisilla tavoilla, eikä niiden toimintaa voi suoraan verrata toisiinsa. Kuitenkin olemme myös osoittaneet, että näiden kolmen järjestelmän määrittelemien laskettavien funkitoiden joukot ovat yhtäpitävät. Tämä tarkoittaa sitä, että mikäli voimme laskea jonkin funktion yhdellä edellä esitellyistä tavoista, voimme laskea sen myös millä tahansa muulla tavalla. Laskettavien funktioiden joukkojen yhtäpitävyys ei kuitenkaan rajoitu vain näihin kolmeen lähestymistapaan. Edellä käsiteltyjen laskettavuuden määritelmien lisäksi laskettavuuden käsitettä ovat omilla tavoillaan määritelleet ainakin Church (1936), Post (1943) ja Markov (1951) ja voidaan osoittaa, että kaikkien näiden (sekä muidenkin tunnettujen laskettavuuden määritelmien) lähestymistapojen avulla määritellyt laskettavien funktioiden joukot ovat yhtäpitävät.

Helposti syntyy halu kysyä, antavatko kaikki mahdolliset laskettavuuden määritelmät tulokseksi vastaavan laskettavien funktioiden joukon? Tätä kysymystä pohtivat ainakin Turing ja Church määriteltyään omat laskettavuuden käsitteensä ja molemmat päätyivät tahoillaan siihen lopputulokseen, että heidän määrittelemänsä koneet pystyisivät laskemaan kaiken, mikä voitaisiin algoritmisesti laskea. Tämä johtopäätös voidaan esittää täsmällisesti ns. *Churchin teesinä* (tunnetaan myös *Church-Turingin teesinä*). Teesi voidaan muotoilla seuraavalla tavalla (teesissä Turing-laskettavuus olisi toki mahdollista korvata jollakin muulla tunnetulla laskettavuuden määritelmällä):

Churchin teesi:
Jos funktio on algoritmisesti laskettava, on se myös Turing-laskettava.

Churchin teesi ei siis ole lause, sillä sitä ei ole mahdollista osoittaa todeksi. Väite on kyllä mahdollista todistaa vääräksi esimerkiksi esittämällä intuitiivisesti laskettava funktio, jota ei kuitenkaan ole mahdollista laskea Turingin

koneella. Tätä ei kuitenkaan pidetä kovin todennäköisenä, sillä Churchin teesillä on vahvaa näyttöä puolellaan.

Ensinnäkin kaikki laskettavuuden tunnetut määritelmät antavat saman laskettavien funktioiden joukon. Tämä joukko on myös sen kokoinen, että kaikki siinä laskettavat funktiot voidaan todeta intuitiivisesti laskettavaksi. Toiseksi väitettä ei ole yrityksistä huolimatta onnistuttu kumoamaan. Tämä ei kuitenkaan todista sitä, etteikö väitettä voitaisi todistaa. Kolmanneksi algoritmin käsite tukee teesiä, sillä vaikuttaa mahdottomalta, että olisi olemassa funktio, joka voidaan laskea tiettyä algoritmia käyttäen, mutta jota ei voida laskea Turingin koneella tai rekursiivisesti askel askeleelta.

Viitteet

[1] Cutland, Nigel. *Computability*, Cambridge University Press, 2000

[2] Odifreddi, Piergiorgio. *Classical Recursion Theory*, Eleviser Science Publishishers B. V., 1989

[3] Wood, Deric. *Theory of computation*, Harper & Row, Publishers, Inc., 1987