Ohjelmointikielen PCF todistusjärjestelmä

Saara Huhmarniemi

22 maaliskuuta 2000

1 Johdanto

Ohjelmointikielellä laaditun ohjelman suoritus määritellään yleensä antamalla ohjelmointikielelle kääntäjä. Ohjelman suoritus voidaan myös määritellä matemaattisesti, todistusjärjestelmän avulla. Todistusjärjestelmä sisältää tarvittavat säännöt ohjelmointikielen lauseiden muodostamista sekä ohjelman tuloksen laskemista varten. Ohjelman tulos voidaan laskea joko evaluoimalla ohjelmaa kuvaavaa lauseketta tai suorittamalla sarja päättelyaskeleita. Evaluointi voidaan myös suorittaa esimerkiksi laskentaa suorittavan koneen tilojen avulla.

Ohjelmointikielen PCF (Programming Computable Functions) suoritus on määritelty todistusjärjestelmänä. Luvussa 2 tarkastellaan PCF-kielisen ohjelman aksiomaattista- ja suoritussemantiikkaa, jotka muodostavat yhdessä todistusjärjestelmän. Lisäksi ohjelmointikielellä on denotationaalinen semantiikka, joka käsittelee ohjelman malliteoreettista tulkintaa. Suoritussemantiikasta käsitellään tarkemmin rekursiota ja sen käyttöä osittain rekursiivisten funktioiden määrittelyssä.

Funktionaalisen ohjelman suorituksen voidaan ajatella olevan ohjelman lausekkeiden sieventämistä määrättyjen sääntöjen mukaan. Ohjelman suoritustapa vaihtelee riippuen siitä, missä järjestyksessä ohjelman alilausekkeita sievennetään. PCF:n sievennysjärjestys on vasemmanpuoleinen tai laiska. Samaa sievennysjärjestystä käytetään myös muissa puhtaasti funktionaalisissa ohjelmointikielissä kuten Haskell ja Miranda. Esimerkiksi ohjelmointikielessä Lisp sekä monissa käytännön ohjelmointikielissä on käytössä ahne sievennys. Eri strategioiden eroja sekä etuja ja heikkouksia pyritään valaisemaan luvussa 5.

Esitelmä perustuu lähes täysin John C. Mitchellin kirjan Foundations for Programming Languages [Mit96] lukuun 2 ja etenkin alilukuun 2.4.

2 Ohjelmointikieli PCF

PCF on tyypitetty funktionaalinen ohjelmointikieli ja sen perusteet ovat λ -laskennassa. Kieli on suunniteltu nimenomaan ohjelmointikielten analysointia varten, mutta sen avulla on mahdollista kirjoittaa pieniä funktionaalisia ohjelmia suhteellisen helposti. PCF:n ilmaisuvoimasta kertoo se, että kaikki *osittain rekursiiviset* funktiot ovat määriteltävissä siinä. Yleensä uskotaan, että kaikki mekaanisesti laskettavat funktiot ovat osittain rekursiivisia.

2.1 PCF:n syntaksi

Tässä kappaleessa esitetään PCF:n syntaksi niiltä osin, mitä tarvitaan esitettävien esimerkkiohjelmien laatimiseen. Myöhemmin esitetään lisäksi tarkemmat lausekkeenmuodostussäännöt. Jokaisella PCF:n lausekkeella on yksikäsitteinen tyyppi. Kielen perustyypit ovat:

luonnolliset luvut: nat $(0, 1, 2, ...)$

totuusarvot: *bool* $(true, false)$

funktiot: $\sigma \to \tau$, missä σ on funktion lähtöjoukko ja τ arvojoukko.

Lisäksi ohjelmoijan on mahdollista esitellä uusia tyyppejä. Uusien tyyppien määritteleminen kuitenkin ohitetaan tässä yhteydessä.

Kieli sisältää muuttujia $x, y, z, \ldots$ Muuttujan tyyppi määrätään muuttujan esittelyssä:

$\lambda x : \sigma$ tai $x_1 : \sigma_1, ..., x_k : \sigma_k$

PCF:n lausekkeita ovat mm. seuraavat:

Ehtolausekkeet:

if $\langle$*bool_lauseke*$\rangle$ then $\langle$*bool_lauseke*$\rangle$ else $\langle$*bool_lauseke*$\rangle$
if $\langle$*bool_lauseke*$\rangle$ then $\langle nat\_lauseke \rangle$ else $\langle nat\_lauseke \rangle$

Yhteenlasku:

$\langle nat\_lauseke \rangle + \langle nat\_lauseke \rangle$

Vertailu:

Eq? $\langle nat\_lauseke \rangle \langle nat\_lauseke \rangle$, joka on tyyppiä *bool*.

3 PCF todistusjärjestelmänä

PCF:n ohjelmien todistukset tehdään formaalissa järjestelmässä, johon kuuluvat primitiivitason objektit: muuttujat, atomikaavat ja lausekkeiden muodostussäännöt.

PCF:n lausekkeita merkitään isoilla kirjaimilla $(M, N, P, \ldots)$. Lausekkeita kutsutaan etenkin päättelyjärjestelmien yhteydessä termeiksi, ja niihin sisältyviä alilausekkeita alitermeiksi. Lauseketta, joka alkaa λ-abstraktiolla kutsutaan toisinaan funktioksi. Luonnollisia lukuja merkitään pienillä kirjaimilla $(m, n, \ldots)$. Päättelysäännöt esitetään muodossa

$$\frac{\Psi_1, \Psi_2, \ldots}{\Theta}$$

mikä tarkoittaa, että oletuksista $\Psi_1, \Psi_2, \ldots$ voidaan päätellä Θ.

3.1 Aksiomaattinen semantiikka

Aksiomaattinen semantiikka sisältää todistusjärjestelmän aksioomat ohjelman ja sen osien ominaisuuksille. Ominaisuus voi olla yhtälö tai väite, joka koskee ohjelman tulosta tietyillä syötteillä, tai jotain muuta. Aksiomaattinen semantiikka siis määrittelee ohjelman tuloksen, jos ohjelmalla on sellainen. PCF:n todistusjärjestelmässä todistettavat ominaisuudet ovat yhtälöitä. Niinpä kaksi ohjelmaa ovat aksiomaattiselta semantiikaltaan ekvivalentit, kun niistä on johdettavissa täsmälleen samat väitteet. Lisäksi kaksi aksiomaattiselta semantiikaltaan ekvivalenttia lauseketta on mahdollista korvata toisillaan siten, että ohjelman suoritussemantiikka ei muutu.

Aluksi määritellään aksioomat, joiden avulla voidaan muodostaa ohjelmointikieleen PCF kuuluvia lausekkeita.

Yhtäsuuruus	
(ref)	$M = M$
Tyypit *nat* ja *bool*	
(add)	$0+0=0, 0+1=1, \ldots, 3+5=8, \ldots$ [1]
$(Eq?)$	$\mathrm{Eq?}\, n\, n = true, \quad \mathrm{Eq?}\, n\, m = false, \qquad m \neq n$ lukuja
$(cond)$	$\mathrm{if}\, true\, \mathrm{then}\, M\, \mathrm{else}\, N = M, \qquad \mathrm{if}\, false\, \mathrm{then}\, M\, \mathrm{else}\, N = N$
Sidonta	
(α)	$\lambda x : \sigma.M = \lambda y : \sigma.[y/x]M, \qquad y$ ei esiinny vapaana M:ssä
Funktiot	
(β)	$(\lambda x : \sigma.M)N = [N/x]M$
(η)	$\lambda x : \sigma.Mx = M, \qquad x$ ei esiinny vapaana M:ssä
Rekursio	
(fix)	$fix_\sigma = \lambda f : \sigma \to \sigma.f(fix_\sigma f)$

PCF:n aksiomaattiseen semantiikkaan kuuluvat päättelysäännöt (inference rules) on koottu alla. Päättelysääntöjä tarvitaan, kun kahta lauseketta verrataan toisiinsa tarkoituksena selvittää onko kyseessä sama lauseke. Päättelysäännöt on muotoiltu siten, että kahden lausekkeen samuus on ekvivalenssirelaatio. Tällöin lausekkeiden vertailussa voidaan käyttää hyväksi ekvivalenssirelaation ominaisuuksia (symmetrisyys, transitiivisuus, jne.).

Ekvivalenssi
$(symm)$, $(trans)$

$$\frac{M = N}{N = M} \qquad \frac{M = N N = P}{M = P}$$

Kongruenssi
Tyypit *nat* ja *bool*

$$\frac{M = N, P = Q}{M + P = N + Q} \qquad \frac{M = N, P = Q}{\mathrm{Eq?}\, M\, P = \mathrm{Eq?}\, N\, Q}$$

$$\frac{M_1 = M_2, N_1 = N_2, P_1 = P_2}{\mathrm{if}\, M_1\, \mathrm{then}\, N_1\, \mathrm{else}\, P_1 = \mathrm{if}\, M_2\, \mathrm{then}\, N_2\, \mathrm{else}\, P_2}$$

Aksioomien perusteella voidaan muodostaa esimerkkilausekkeet (1) ja (2), joista jälkimmäinen on ohjelma, joka laskee luvulle n kertoman ($n!$).

$$\lambda x : nat.\, \mathrm{if}\ \mathrm{Eq?}\, x\, 2\, \mathrm{then}\, x\, \mathrm{else}\, x + 2 \tag{1}$$

$$fact : nat \to nat = \lambda x : nat.\, \mathrm{if}\ \mathrm{Eq?}\, x\, 1\, \mathrm{then}\, 1\, \mathrm{else}\, x * fact(x - 1) \tag{2}$$

3.2 Suoritussemantiikka

Ohjelman suoritussemantiikka määrittelee todistusjärjestelmän päättelysäännöt, joiden avulla ohjelmaa kuvaavaa lauseketta on mahdollista evaluoida askel askeleelta. Määritellään PCF:lle reduktioaksioomat, jotka määräävät, miten PCF-ohjelma suoritetaan. Reduktioaksioomat esitetään siten, että jos lausekkeesta M voidaan johtaa lauseke N, merkitään $M \to N$. Aksiomaattisen semantiikan aksioomiin verrattuna reduktioaksioomilla on suunta, joka vastaa ohjelman suorituksen etenemissuuntaa. Reduktioaksioomat sisältyvät aksiomaattisessa semantiikassa määriteltyihin yhtälöihin. Reduktioaksioomissa ei ole otettu kantaa siihen, missä järjestyksessä päättelysääntöjä sovelletaan tai mitä ohjelman osaa evaluoidaan ensimmäiseksi.

Tyypit nat ja $bool$	
(add)	$0 + 0 \to 0, 0 + 1 \to 1, ..., 3 + 5 \to 8, ...$
$(Eq?)$	$\mathrm{Eq?}\, n\, n \to true \quad \mathrm{Eq?}\, n\, m \to false, \quad m \neq n$ lukuja
$(cond)$	$\mathrm{if}\, true\, \mathrm{then}\, M\, \mathrm{else}\, N \to M, \quad \mathrm{if}\, false\, \mathrm{then}\, M\, \mathrm{else}\, N \to N$
Sidottujen muuttujien uudelleennimeäminen	
(α)	$\lambda x : \sigma.M \to \lambda y : \sigma.[y/x]M, \quad y$ ei esiinny vapaana M:ssä
Funktiot$(\sigma \to \tau)$	
(β)	$(\lambda x : \sigma.M)N \to [N/x]M$
Rekursio	
(fix)	$fix_\sigma \to \lambda f : \sigma \to \sigma.f(fix_\sigma f)$

Esimerkkilauseke (1) sievennettynä.

$$\begin{aligned} & (\lambda x : nat.\, \mathrm{if}\ \mathrm{Eq?}\, x\, 2\, \mathrm{then}\, x\, \mathrm{else}\, x + 2)1 \\ \to_{(\beta)} \quad & \mathrm{if}\ \mathrm{Eq?}\, 1 2\, \mathrm{then}\, 1\, \mathrm{else}\, 1 + 2 \\ \to_{(cond)} \quad & 1 + 2 \\ \to_{(add)} \quad & 3 \end{aligned}$$

4 Rekursio PCF:ssä

Lasketaan luvun 3 kertoma 3! funktion (2) avulla.

$$\begin{array}{ll} & fact\, 3 \\ = & (\lambda x : nat.\, \text{if Eq? } x\, 1 \text{ then } 1 \text{ else } x * fact(x-1))3 \\ \rightarrow_{(\beta)} & (\text{ if Eq? } 3\, 1 \text{ then } 1 \text{ else } 3 * fact(3-1)) \\ \rightarrow_{(Eq?)} & 3 * fact(3-1)) \\ \rightarrow & 3 * fact\, 2 \end{array}$$

Tuloksena saadaan lauseke, jossa tulee suorittaa uudelleen funktion $fact$ 'kutsu'. Sijoitetaan funktion lauseke muuttujan $fact$ tilalle, ja lasketaan edelleen.

$$\begin{array}{ll} & 3 * fact\, 2 \\ = & 3 * (\lambda x : nat.\, \text{if Eq? } x\, 1 \text{ then } 1 \text{ else } x * fact(x-1))2 \\ \rightarrow_{(\beta)} & 3 * \text{ if Eq? } 2\, 1 \text{ then } 1 \text{ else } 2 * fact(2-1) \\ \rightarrow_{(Eq?)} & 3 * 2 * fact(2-1) \\ \rightarrow & 3 * 2 * fact\, 1 \\ = & 3 * 2 * (\lambda x : nat.\, \text{if Eq? } x\, 1 \text{ then } 1 \text{ else } x * fact(x-1))1 \\ \rightarrow_{(Eq?)} & 3 * 2 * \text{ if Eq? } 1\, 1 \text{ then } 1 \text{ else } 1 * fact(1-1) \\ \rightarrow_{(\beta)} & 3 * 2 * 1 \\ = & 6 \end{array}$$

Tarkastellaan alkuperäisen funktion (2) rekursiivista määritelmää. Muutetaan funktio yhtälön muotoon, jolloin $fact$ on ratkaisu yhtälöön

$$fact = \lambda x : nat.\, \text{if Eq? } x\, 1 \text{ then } 1 \text{ else } x * fact(x-1) \tag{3}$$

Muuttuja $fact$ esiintyy nyt yhtälössä kaksi kertaa, sekä yhtälön vasemmalla että oikealla puolella. Koska $fact$ esiintyy vapaana yhtäsuuruusmerkin oikealla puolella, muuttujaa $fact$ ei voida asettaa suoraan viittaamaan oikealla puolella olevaan lausekkeeseen. Matemaattiselta kannalta ei ole nimittäin varmaa, että yhtälöllä muotoa $f : \sigma = M$, missä M on mielivaltainen PCF:n lauseke, joka sisältää f:n olisi aina ratkaisu (esim. yhtälöllä $x = x + 1$ ei ole ratkaisua). Yhtälöllä voi toisaalta myös olla useita mahdollisia ratkaisuja.

Sen sijaan, että etsittäisiin muuttujalle $fact$ ratkaisua yhtälöstä (3), tavoitteena on että muuttuja $fact$ viittaa *rekursiivisesti* funktioon, joka on määritelty lausekkeella. Rekursiivisuuden ilmaisemista varten otetaan käyttöön uusi PCF:n perusfunktio, *kiintopisteoperaattori*. Kiintopisteoperaattorin avulla voidaan varmistaa, että rekursiivinen lauseke on määritelty.

Yleisesti, jos $F : \sigma \rightarrow \sigma$ on funktio joltain tyypiltä itselleen, on F:n kiintopiste (fixed-point) arvo x s.e. $F(x) = x$. Kiintopisteoperaattoria merkitään fix_σ, jokaista tyyppiä σ kohden. Kiintopisteen avulla voidaan PCF:ssä määritellä funktioiden perhe $fix_\sigma : (\sigma \rightarrow \sigma) \rightarrow \sigma$, jokaiselle tyypille σ. Kiintopisteoperaattorin evaluointisääntö (fix) on annettu reduktioaksioomissa viimeisenä.

Käyttämällä λ-abstraktiota saadaan rekursiivinen määritelmä (3) muutettua ei-rekursiiviseen muotoon.

$$fact = \underbrace{(\lambda g : nat \to nat.\lambda x : nat.\, \text{if Eq?}\, x\, 1 \,\text{then}\, 1 \,\text{else}\, x * g(x-1))}_{M} fact \tag{4}$$

Muodostetaan kiintopisteoperaattorin fix avulla funktion M kiintopiste.

$$fact = fix_{nat \to nat} M \tag{5}$$

Nyt on mahdollista käyttää reduktioaksioomien sääntöä (fix). λ-abstraktion avulla sääntö voidaan muuttaa muotoon, jota on mahdollista sieventää edelleen.

$$\begin{array}{ll} & \underline{fix_{nat \to nat} M} \\ \to_{(fix)} & \underline{((\lambda h : (nat \to nat) \to (nat \to nat).h(fix_{nat \to nat} h))M)} \\ \to_{(\beta)} & (M(fix_{nat \to nat} M)) \end{array} \tag{6}$$

Lasketaan kiintopisteoperaatiota käyttäen funktion $fact$ arvo luvulle n. Operaattorin fix tyyppi $nat \to nat$ jätetään käytännön syistä merkitsemättä.

$$\begin{array}{ll} & fact\, n \\ =_{(5)} & (fix M)n \\ \to_{(6)} & (M(fix M))n \\ = & ((\lambda g : nat \to nat.\lambda x : nat.\, \text{if Eq?}\, x\, 1 \,\text{then}\, 1 \,\text{else}\, x * g(x-1))(fix M))n \\ \to_{(\beta)} & (\lambda x : nat.\, \text{if Eq?}\, x\, 1 \,\text{then}\, 1 \,\text{else}\, x * (fix M)(x-1))n \\ \to_{(\beta)} & (\,\text{if Eq?}\, n\, 1 \,\text{then}\, 1 \,\text{else}\, n * (fix M)(n-1)) \end{array} \tag{7}$$

Näin saadusta muodosta voidaan nähdä, että kun $n = 1$, tuloksena saadaan 1. Kun $n > 1$, on lauseketta $(fix M)(n-1)$ mahdollista sieventää edelleen samaan tapaan kuin kohdassa (7). Käyttämällä kiintopisteoperaattoria fix, funktion $fact$ avulla on mahdollista laskea kertoma $n!$ jokaiselle luonnolliselle luvulle n, kun $n > 0$. Sen lisäksi funktion oikeellisuus on myös mahdollista todistaa induktiivisesti. Kiintopisteoperaattorin käyttö on itse asiassa induktioperiaatteen soveltamista laskettavaan lausekkeeseen.

Kiintopisteoperaattori siis sallii numeeristen funktioiden määrittelyn PCF:ssä rekursion avulla. Rekursiota käyttäen on kuitenkin mahdollista muodostaa funktioita, jotka eivät ole määriteltyjä koko arvoalueellaan. PCF:ssä tyyppiä σ oleva lauseke siis määrittelee algoritmin, joka laskee tyyppiä σ olevan elementin arvon. On mahdollista, että lausekkeen suoritusalgoritmi ei päädy mihinkään tyypin σ standardiarvoon, jolloin algoritmin suoritus ei pysähdy lainkaan. Esimerkissä (8) on funktiosta $fact$ muodostettu pysähtymätön versio, jossa funktion argumenttia kasvatetaan sen vähentämisen sijaan. Tällöin ohjelman suoritus ei pysähdy, kun x:n arvo on suurempi kuin 1.

$$fact_2 = \lambda x : nat.\, \text{if Eq?}\, x\, 1 \,\text{then}\, 1 \,\text{else}\, x * fact_2(x+1) \quad \text{kun } x > 1 \tag{8}$$

Kiintopisteoperaattorin käyttö ei edellytä, että määriteltäisiin, milloin operaattorin soveltaminen lopetetaan. Operaattori fix korvataan ohjelmointikielissä usein käytännön syistä jollain muulla konstruktiolla (esimerkiksi erilaisilla silmukoilla) jotka eivät salli rekursiivisten algoritmien määrittelyä. Tällöin vältytään tilanteilta, joissa ohjelman suoritus ei pysähdy, mutta samalla menetetään keino määritellä rekursiivisia funktioita.

5 Sievennysstrategiat

PCF-kielen reduktioaksioomat eivät ota kantaa siihen, missä järjestyksessä lausekkeen osat sievennetään. Periaatteessa jokaiselle hieman monimutkaisemmalle PCF:n termille M on olemassa useita eri sievennyspolkuja ($M \rightarrow N_1 \rightarrow N_2 \rightarrow \cdots$), joista osa päätyy normaalimuotoon $F(M)$, jos sellainen on olemassa. Osatermien sievennysjärjestystä rajoittamalla voidaan muodostaa deterministisiä sievennysstrategioita, joiden soveltamisen tuloksena syntyneet sievennyspolut eroavat toisistaan. Strategioiden periaatteelliset erot tulevat näkyviin, kun tarkastellaan erilaisia vaihtoehtoisia tapoja sieventää muotoa $(\lambda x : \sigma.M)N$ olevaa lauseketta, kun $M \rightarrow M'$ sekä $N \rightarrow N'$.

1. $(\lambda x : \sigma.M)N \rightarrow [N/x]M$ $\quad$ (*vasemmanpuoleinen/laiska*)
2. $(\lambda x : \sigma.M)N \rightarrow (\lambda x : \sigma.M)N'$ $\quad$ (*ahne*)
3. $(\lambda x : \sigma.M)N \rightarrow (\lambda x : \sigma.M')N$ $\quad$ (*optimointi*)

Kolmas vaihtoehto on oikeastaan koodin optimointia ennen sijoituksen suorittamista. Keskitytään tarkastelemaan kahteen ensimmäiseen periaatteeseen perustuvia strategioita.

Ensimmäisenä on esitetty *vasemmanpuoleinen* (left-most) -sievennys. Sen periaatteena on edetä vasemmalta oikealle ja suorittaa λ-abstraktioiden sovellukset siinä järjestyksessä kun ne tulevat vastaan. Lauseke ei suinkaan aina sananmukaisesti 'sievene', etenkin jos sijoitettava lauseke N on monimutkainen ja esiintyy M:ssä useita kertoja. Sama periaate on käytössä myös *laiskassa* (lazy) sievennyksessä.

Vasemmanpuoleinen sievennysstrategia käy läpi termin sieventäen jokaisen sen alilausekkeen, ja löytäen termille normaalimuodon $F(M)$ aina, kun sellainen on olemassa. Koska laiskaa sieventäminen johtaa lähes aina samaan lopputulokseen kuin vasemmanpuoleinen, tarkastellaan tarkemmin vain laiskan sievennysstrategian aksioomia, sekä lisäksi joitain vasemmanpuoleisen ja laiskan sievennysstrategian eroja. Kolmatta sievennysstrategiaa, joka pyrkii sieventämään sijoitettavaa lauseketta ennen β-reduktiota, kutsutaan *ahneeksi* (eager) sievennykseksi.

5.1 Laiska sievennysstrategia

Laiskassa sievennysstrategiassa siis edetään lauseketta vasemmalta oikealle sieventäen alilausekkeita. Strategian tavoitteena ei kuitenkaan ole lausekkeen sieventäminen aina normaalimuotoon asti, kuten vasemmanpuoleisessa sievennyksessä, vaan pikemminkin 'tarpeen mukaan'. Laiska sievennysstrategia määritellään seuraavien aksioomien avulla.

1. Aksioomat

$$\frac{M \rightarrow N}{M \rightarrow N} \qquad M \rightarrow N \text{ on reduktioaksiooma}$$

2. Alitermien päättelysäännöt
   (nat ja $bool$)

$$\frac{M \to M'}{M + N \to M' + N} \qquad \frac{M \to M'}{n + M \to n + M'} \quad n \text{ on luku}$$

$$\frac{M \to M'}{\text{Eq?}\, M\, N \to \text{Eq?}\, M'\, N} \qquad \frac{M \to M'}{\text{Eq?}\, n\, M \to \text{Eq?}\, n\, M'} \quad n \text{ on luku}$$

   Yhteenlaskun sekä vertailulausekkeen vasemmanpuoleiset argumentit sievennetään lukuarvoihinsa ennen oikeanpuoleisten argumenttien sieventämistä.

$$\frac{M \to M'}{\text{if}\, M\, \text{then}\, N\, \text{else}\, P \to \text{if}\, M'\, \text{then}\, N\, \text{else}\, P}$$

   Ehto-osan lauseke sievennetään ennen kummankaan suoritettavan lausekkeen sieventämistä, jolloin ensin selvitetään, kumpaan ehtolausekkeen haaraan sieventämisessä edetään.

   (funktiot)

$$\frac{M \to M'}{M \to M' N}$$

   Funktiolauseke sievennetään ennen siihen sovellettavan lausekkeen sieventämistä.

Laiska sievennys esimerkkifunktiolle (2). Lasketaan kertoma lausekkeelle $((\lambda z : nat.z)3)$.

$$\begin{aligned}
& fact\,((\lambda z : nat.z)3) \\
= \; & \underline{(fixM)}((\lambda z : nat.z)3) \\
= \; & M(fixM)(\lambda z : nat.z + 1)3) \\
= \; & \underline{(\lambda g : nat \to nat.\lambda x : nat.\, \text{if Eq?}\, x\, 1\, \text{then}\, 1\, \text{else}\, x * g(x-1))(fixM)}((\lambda z : nat.z)3) \\
= \; & \underline{(\lambda x : nat.\, \text{if Eq?}\, x\, 1\, \text{then}\, 1\, \text{else}\, x * (fixM)(x-1))((\lambda z : nat.z)3)} \\
= \; & (\, \text{if Eq?}\, \underline{((\lambda z : nat.z)3)}1\, \text{then}\, 1\, \text{else}\, ((\lambda z : nat.z)3) * (fixM)(((\lambda z : nat.z)3) - 1) \\
= \; & \underline{\text{if Eq?}\, 3 1\, \text{then}\, 1\, \text{else}\, ((\lambda z : nat.z)3) * (fixM)(((\lambda z : nat.z)3) - 1)} \\
= \; & \underline{((\lambda z : nat.z)3)} * (fixM)(((\lambda z : nat.z)3) - 1) \\
= \; & 3 * \underline{(fixM)}(((\lambda z : nat.z)3) - 1) \\
= \; & 3 * M(fixM)(((\lambda z : nat.z)3) - 1) \\
& \ldots
\end{aligned}$$

Laiskan ja vasemmanpuoleisen sievennyksen ero tulee näkyviin lausekkeissa, joiden määrittelemille funktioille ei ole annettu kaikkia arvon laskemiseen vaadittavia parametreja. Tällöin lauseke ei sievene 'loppuun asti', vaan jää yhä funktion muotoon, kuten esimerkiksi lauseke (9).

$$\lambda y : nat.y + 12 \tag{9}$$

Laiskassa sievennystrategiassa tällaista muotoa oleva lauseke katsotaan käyttäjän kannalta yhtä tarpeettomiksi kuin sama lauseke vielä hieman sotkuisemmassa muodossa. Sieventäminen lopetetaan, kun käy ilmi, että lausekkeen lopullinen muoto on funktio. Laiska sievennys on periaatteeltaan samanlainen strategia kuin vasemmanpuoleinen, mutta se pysähtyy λ-abstraktioon, jolle ei löydy sovellettavaa arvoa, kun taas vasemmanpuoleinen jatkaa alitermien sieventämistä aina normaalimuotoon asti (jos sellainen on olemassa). Tilanne, jossa laiskan ja vasemmanpuoleinen sievennysstrategian erot tulevat näkyviin on siis muotoa

$$\lambda x.M, \qquad \text{missä } M \text{ on PDF:n lauseke} \tag{10}$$

Sievennetään esimerkkilauseke ensin vasemmanpuoleista ja sen jälkeen laiskaa sievennysstrategiaa käyttäen.

$$\begin{array}{rl} & (\lambda x : nat.\lambda y : nat.y + x)((\lambda z : nat.z + 4)8) \\ \rightarrow_{vasen} & \lambda y : nat.y + (\lambda z : nat.z + 4)8 \\ \rightarrow_{vasen} & \lambda y : nat.y + (4 + 8) \\ \rightarrow_{vasen} & \lambda y : nat.y + 12 \end{array}$$

$$\begin{array}{rl} & (\lambda x : nat.\lambda y : nat.y + x)((\lambda z : nat.4 + z)8) \\ \rightarrow_{laiska} & \lambda y : nat.y + (\lambda z : nat.4 + z)8 \end{array}$$

5.2 Ahne sievennysstrategia

Ahne sievennysstrategia perustuu siihen, että lauseke sievennetään arvoonsa ennenkuin se voidaan sijoittaa johonkin toiseen lausekkeeseen. Sievennysstrategian soveltamista varten on määriteltävä erityinen *arvon* (value) käsite, jonka avulla voidaan määrätä, milloin lauseke on 'valmis'. Vakiot, muuttujat ja λ-abstraktiot ovat arvoja, mutta lisäksi arvoihin täytyy lisätä kiintopisteoperaattorin fix soveltamisen seurauksena syntynyt lauseke. Jos nimittäin funktion argumenttina olevaa kiintopisteoperaattoria yritetään sieventää edelleen, esimerkin (11) kaltaiset tilanteet ovat mahdollisia.

$$\begin{array}{rl} & fact\, n \\ = & \underline{(fixM)}n \\ \rightarrow_{(6)} & (M\underline{(fixM)})n \\ \rightarrow & ((M)(M\underline{(fixM)}))n \\ & \cdots \end{array} \tag{11}$$

1. Arvot

   1. V on *arvo*, jos V on vakio, muuttuja tai lambda-abstraktio $(\lambda x : \sigma.M)$
   2. $delay_{\sigma\rightarrow\tau}[M] = \lambda x : \sigma.Mx, \qquad x$ ei vapaa M:ssä

2. Aksioomat

Tyypit *nat* ja *bool*	
(add)	$0+0 \to 0, 0+1 \to 1, ..., 3+5 \to 8, ...$
$(Eq?)$	$\text{Eq?}\, n\, n \to true, \quad \text{Eq?}\, n\, m \to false, \quad m \neq n$ lukuja
$(cond)$	$\text{if}\, true\, \text{then}\, M\, \text{else}\, N \to M, \quad \text{if}\, false\, \text{then}\, M\, \text{else}\, N \to N$
Funktiot	
(β)	$(\lambda x : \sigma.M)V \to [V/x]M, \quad V$ on arvo
Rekursio	
(fix)	$fix_{\sigma\to\tau} V \to V(delay_{\sigma\to\tau}[fix_{\sigma\to\tau} V]), \quad V$ on arvo

3. Alitermien päättelysäännöt

Tyyppien (nat) ja $(bool)$ osalta alitermien päättelysäännöt ovat samat kuin laiskassa sievennyksessä. Ahneen strategian ero laiskaan nähden tulee esiin funktioiden sievennykseen liittyvissä päättelysäännöissä.

(funktiot)

$$\frac{M \to M'}{M \to M'N} \qquad \frac{N \to N'}{VN \to VN'} \qquad V \text{ on arvo}$$

Päättelysäännöt merkitsevät, että funktion lauseke sievennetään arvoonsa ennen funktion argumentin sieventämistä. (Myös λ-abstraktio on siis määritelty arvoksi.) Tämän jälkeen funktion argumentti sievennetään ennen sen soveltamista funktiolausekkeeseen.

Ahne reduktio esimerkkifunktiolle (2).

$$\begin{aligned}
& fact\,((\lambda z : nat.z)3) \\
= \; & \underline{(fix M)}((\lambda z : nat.z)3) \\
= \; & (M(delay[fix M]))((\lambda z : nat.z)3) \\
= \; & \underline{((\lambda g : nat \to nat.\lambda x : nat.\,\text{if Eq?}\, x\, 1\, \text{then}\, 1\, \text{else}\, x * g(x-1))(delay[fix M]))}((\lambda z : nat.z)3) \\
= \; & (\lambda x : nat.\,\text{if Eq?}\, x\, 1\, \text{then}\, 1\, \text{else}\, x * (delay[fix M])(x-1))\underline{((\lambda z : nat.z)3)} \\
= \; & \underline{(\lambda x : nat.\,\text{if Eq?}\, x\, 1\, \text{then}\, 1\, \text{else}\, x * (delay[fix M])(x-1))3} \\
= \; & \underline{\text{if Eq?}\, 3\, 1\, \text{then}\, 1\, \text{else}\, 3 * (delay[fix M])(3-1)} \\
= \; & 3 * (delay[fix M])(3-1) \\
= \; & 3 * (\lambda y : nat.(fix M)y)(3-1) \\
& \ldots
\end{aligned}$$

Ahneen sievennysstrategian avulla on toisinaan mahdollista saavuttaa lyhyempi sievennyspolku kuin laiskaa sievennystä käyttämällä. Muotoa $(\lambda x : \sigma.M)N$ olevilla lausekkeilla, joissa lauseke N ei esiinny lausekkeessa M, ahne sievennysstrategia kuitenkin vaatii enemmän töitä kuin laiska sievennysstrategia. Laiskaa strategiaa käyttäen lauseke sievenee aina samalla tavoin:

$$\begin{aligned}
& (\lambda x : \sigma.M)N \\
\to_{laiska} \; & \lambda x : \sigma.M
\end{aligned}$$

Kun taas ahne strategia sieventää funktion argumenttia N, kun $N \to N'$.

$$\begin{array}{rl} & (\lambda x : \sigma.M)N \\ \to_{ahne} & (\lambda x : \sigma.M)N' \\ & \ldots \end{array}$$

Mikäli termillä N ei ole lainkaan normaalimuotoa, ahne sievennysstrategia jatkaa lausekkeen N sievennystä loputtomiin. Esimerkkinä päättymättömästä laskennasta on lausekkeen (8) soveltaminen lausekkeeseen $(\lambda z : nat.1)$, ensin ahnetta strategiaa käyttäen.

$$\begin{array}{rl} & (\lambda z : nat.1)(((\lambda g : nat \to nat.\lambda x : nat.\,\mathrm{if}\ \mathrm{Eq?}\, x\, 1\, \mathrm{then}\, 1\, \mathrm{else}\, x * g(x+1))(fixM)2) \\ \to_{ahne} & (\lambda z : nat.1)(2 * 3 * 4 * \cdots) \end{array}$$

Laiska sievennys samalle lausekkeelle

$$\begin{array}{rl} & (\lambda z : nat.1)(((\lambda g : nat \to nat.\lambda x : nat.\,\mathrm{if}\ \mathrm{Eq?}\, x\, 1\, \mathrm{then}\, 1\, \mathrm{else}\, x * g(x+1))(fixM))2) \\ \to_{laiska} & 1 \end{array}$$

6 Yhteenveto

Ohjelman suorituksen määritteleminen todistusjärjestelmän avulla vaatii ohjelman aksiomaattisen- ja suoritussemantiikan määrittelemisen. Aksiomaattinen semantiikka sisältää ohjelman lausekkeiden muodostussäännöt ja suoritussemantiikka kuvaa ohjelman suorituksen etenemisen. Funktionaalisen ohjelman suoritusta voidaan ajatella muodostettujen lausekkeiden sieventämisenä. Erilaiset sievennysstrategiat tarjoavat erilaisia ohjelman suoritustapoja. Strategian valinnalla voidaan vaikuttaa ohjelman suoritusnopeuteen ja sen suorittaman laskennan tehokkuuteen. Esimerkiksi ahne sievennysstrategia ei salli rekursion toteuttamista suoraviivaisesti.

Viitteet

[Mit96] Mitchell J.C.: Foundations for Programming Languages. MIT Press, 1996.