hyväksymispäivä arvosana

arvostelija

Tyypinpäättely

Jani Kirmanen

Helsinki 19.3.2003

Seminaari

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Sisältö

1 Johdanto

Näkemämme lambdakalkyylin tyypinpäättelyalgoritmit perustuvat eksplisiittiseen tyyppinotaatioon. Ennen kaikkea ne vaativat lambda-funktion argumentin tyypin merkitsemisen sen yhteyteen. Tässä esitelmässä kehitetään voimakkaampi tyypinpäättelyalgoritmi, joka perustuu tyypin uudelleenrakentamiseen. Tyypin uudelleenrakennusta eli rekonstruktiota voidaan käyttää päättelemään termien tyypittyvyys ilman erillisten tyyppinimien käyttöä.

2 Tyyppimuuttujat ja korvaukset

Tähän saakka olemme olettaneet, että käytettävissämme on ääretön kokoelma tulkitsemattomia perustyyppejä, joille ei ole määritelty operaatioita. Tulkittuja perustyyppejä ovat mm. Nat ja Bool. Tulkitsemattomia perustyyppejä käytetään tyyppimuuttujina, joita voidaan korvata toisilla tyypeillä tai tyyppien ilmentymillä.

Tyyppikorvaus on kuvaus σ tyyppimuuttujalta t tyypille T, $\sigma = [t \to T]$. Tyyppikorvauksen lähtöjoukko on jokaisen parin vasemmanpuoleinen alkio ja maalijoukko oikeanpuoleinen alkio. Sama alkio voi esiintyä tyyppikorvauksessa sekä vasemmalla että oikealla puolella. Tällaisessa tapauksessa kaikki korvaukset suoritetaan yhdellä kertaa.

Tyyppikorvauksen sovellus on määritelty seuraavasti:

$$\sigma(X) = \begin{cases} T, & (X \mapsto T) \in \sigma \\ X, & X \notin dom(\sigma) \end{cases} \tag{1}$$

$$\sigma(\text{Nat}) = \text{Nat} \tag{2}$$

$$\sigma(\text{Bool}) = \text{Bool} \tag{3}$$

$$\sigma(T1 \to T2) = \sigma(T1) \to \sigma(T2) \tag{4}$$

$$\sigma(x1 : T1, \ldots, xn : Tn) = (x1 : \sigma(T1), \ldots, xn : \sigma(Tn)). \tag{5}$$

Jos σ ja γ ovat tyyppikorvauksia, $\sigma \circ \gamma$ on määritelty seuraavasti:

$$\sigma \circ \gamma = \begin{cases} X \mapsto \sigma(T) & \text{jokaiselle}(X \mapsto T) \in \gamma \\ X \mapsto T & \text{jokaiselle} X \mapsto T \in \sigma \notin dom(\gamma) \end{cases} \tag{6}$$

$$(\sigma \circ \gamma)S = \sigma(\gamma S) \tag{7}$$

Oletetaan että termi t sisältää tyyppimuuttujia ja Γ on määrittelyjoukko, joka myös voi sisältää tyyppimuuttujia. Nyt voidaan kysyä ovatko kaikki t:n korvaamalla tuotetut ilmentymät hyvin tyypitettyjä? Eli päteekö kaikille $\sigma\sigma\Gamma \vdash \sigma t : T$ jollekin T? Toisaalta voidaan myös kysyä, onko olemassa jokin korvaamalla tuotettu ilmentymä, jolla $\sigma\Gamma \vdash \sigma t : T$ jollekin T?

Ensimmäisen näkökulman mukaan tyyppimuuttujien tulee pysyä abstrakteina tyyppitarkastuksen aikana. Näin varmistetaan, että hyvin tyypittyvät termit käyttäytyvät oikein huolimatta tehdystä korvauksesta. Tyyppimuuttujien pitäminen abstrakteina johtaa parametriseen polymorfismiin, missä tyyppimuuttujia käytetään piilottamaan se tosiasia, että termiä voidaan käyttää useissa eri konteksteissa eri perustyypeillä.

Toisen näkökulman mukaan alkuperäinen termi t ei välttämättä ole edes hyvin tyypitetty. Haluammekin tietää, voiko kyseinen termi ilmentää *jotain* tyyppiä oikean tyyppikorvauksen avulla.

Validien ilmentymien etsintä johtaa tyypinpäättelyn ideaan, jossa kääntäjä pystyy paikkaamaan ohjelmoijan puutteellisen tyyppeihin liittyvän tiedon. ML:ssä asia on viety niin pitkälle, että mitään tyyppimäärittelyjä ei tarvita. Jäsennyksen aikana jokainen lambda-funktio $\lambda x.t$ merkitään $\lambda x : X.t$, jossa X on jokin tyyppimuuttuja, jota ei ole merkitty mihinkään muuhun lambda-funktioon. Tämän jälkeen tyypinpäättely voidaan suorittaa etsimällä yleisimmät arvot näille muuttujille, jotka toteuttavat termin tyypittyvyyden.

Formalisoidaksemme tyypinpäättelyn tarvitsemme täsmällisen tavan puhua mahdollisista tavoista suorittaa tyyppikorvaus tyyppimuuttujalta tyypille jossain kontekstissa.

Määritelmä: Olkoon Γ konteksti ja t termi. Ratkaisu (Γ, t):lle on pari (σ, T) siten että $\sigma\Gamma \vdash \sigma t : T$.

3 Pakotettu tyypittäminen

Esitämme algoritmin, joka annetusta termistä t ja kontekstista Γ laskee joukon pakotteita, joita ratkaisujen (Γ, t):lle tulee noudattaa. Algoritmin intuitio on perimmältään sama kuin tavallisessa tyypintarkitusalgoritmissa; ainoa ero on, että pakotteiden tarkastamisen sijaan ne vain talletetaan myöhempää käyttöä varten. Esimerkiksi applikaatio $t_1 t_2$, jossa $\Gamma \vdash t_1 : T_1$ ja $\Gamma \vdash t_2 : T_2$, voidaan sen sijaan, että tarkistettaisiin t_1:n olevan muotoa $T_2 \to T_{12}$ ja palautettaisiin T_{12} applikaation tyyppinä, valita uusi muuttuja X, johon sijoitetaan pakote $T_1 = T_2 \to X$ ja palauttaa se (X) applikaation tyyppinä.

Määritelmä: Pakotejoukko C on joukko yhtälöitä $\{S_i = T_i^{i \in 1..n}\}$. Tyyppikorvauksen σ sanotaan yhdistävän yhtälön $S = T$, jos korvausilmentymät σS ja σT ovat identtiset. σ yhdistää (tai noudattaa) C:n jos se yhdistää C:n jokaisen yhtälön.

Määritelmä: Pakotettu tyyppirelaatio $\Gamma \vdash t : T \,|_x C$ on määritelty seuraavasti:

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T \,|_{\emptyset} \{\}} \qquad (\mathrm{CT-VAR})$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2 \,|_{\chi} C}{\Gamma \vdash \lambda x : T_1.t_2 : T_1 \to T_2 \,|_{\chi} C} \qquad (\mathrm{CT-ABS})$$

$$\frac{\begin{array}{c} \Gamma \vdash t_1 \,|_{\chi_1} C_1 \quad \Gamma \vdash t_2 \,|_{\chi_2} C_2 \\ \chi_1 \cap \chi_2 = \chi_1 \cap FV(T_2) = \chi_2 \cap FV(T_1) = \emptyset \\ X \notin \chi_1, \chi_2, T_1, T_2, C_1, C_2, \Gamma, t_1, \mathrm{or}\, t_2 \\ C' = C_1 \cup C_2 \cup \{T_1 = T_2 \to X\} \end{array}}{\Gamma \vdash t_1 t_2 : X \,|_{\chi_1 \cup \chi_2 \cup \{X\}} C'} \qquad (\mathrm{CT-APP})$$

$$\Gamma \vdash 0 : \mathrm{Nat} \,|_{\emptyset} \{\} \qquad (\mathrm{CT-ZERO})$$

$$\frac{\begin{array}{c} \Gamma \vdash t_1 : T \,|_{\chi} C \\ C' = C \cup \{T = \mathrm{Nat}\} \end{array}}{\Gamma \vdash \mathrm{succ}\, t_1 : \mathrm{Nat} \,|_{\chi} C'} \qquad (\mathrm{CT-SUCC})$$

$$\frac{\begin{array}{c}\Gamma \vdash t_1 : T \,|_{\chi} C \\ C' = C \cup \{T = \mathrm{Nat}\}\end{array}}{\Gamma \vdash \mathrm{pred}\, t_1 \,:\, \mathrm{Nat} \,|_{\chi} C'} \qquad (\mathrm{CT-PRED})$$

$$\frac{\begin{array}{c}\Gamma \vdash t_1 : T \,|_{\chi} C \\ C' = C \cup \{T = \mathrm{Nat}\}\end{array}}{\Gamma \vdash \mathrm{iszero}\, t_1 \,:\, \mathrm{Bool} \,|_{\chi} C'} \qquad (\mathrm{CT-ISZERO})$$

$$\Gamma \vdash \mathrm{true} \,:\, \mathrm{Bool} \,|_{\emptyset} \{\} \qquad (\mathrm{CT-TRUE})$$

$$\Gamma \vdash \mathrm{false} \,:\, \mathrm{Bool} \,|_{\emptyset} \{\} \qquad (\mathrm{CT-FALSE})$$

$$\frac{\begin{array}{c}\Gamma \vdash t_1 \,|_{\chi_1} C_1 \quad \Gamma \vdash t_2 \,|_{\chi_2} C_2 \quad \Gamma \vdash t_3 \,|_{\chi_3} C_3 \\ \chi_1, \chi_2, \chi_3 \,\mathrm{nonoverlapping} \\ C' = C_1 \cup C_2 \cup C_3 \cup \{T_1 = \mathrm{Bool}\,, T_2, = T_3\}\end{array}}{\Gamma \vdash \;\mathrm{if}\, t_1 \,\mathrm{then}\, t_2 \,\mathrm{else}\, t_3 \,:\, X \,|_{\chi_1 \cup \chi_2 \cup \{X\}}\, C'} \qquad (\mathrm{CT-IF})$$

Epäformaalisti relaatio $\Gamma \vdash t : T \,|_{\chi}\, C$ voidaan lukea 'termi t on tyyppiä T oletuksilla Γ jos pakotukset C toteutuvat'. Alaindeksi χ:tä käytetään kuvaamaan jokaisessa sovelluksessa esiteltäviä uusia tyyppimuuttujia ja varmistamaan, että jokaisessa vaiheessa luodaan *uusi* tyyppimuuttuja. Kun säännöt luetaan lopusta alkuun, saadaan suoraviivainen toimintaohje, jolla annetusta kontekstista Γ ja termistä t, voidaan laskea tyyppi T ja pakotejoukko C, s.e. $\Gamma \vdash t : T \,|_{x} C$. Osoittautuu, että tällainen algoritmi onnistuu aina löytämään jotkin T ja C, s.e. $\Gamma \vdash t : T \,|_{x} C$. Lisäksi huomataan, että T ja C määräytyvät yksikäsitteisesti Γ:sta ja t:stä.

Idea on, että mielivaltaisesta (Γ, t) voidaan päätellä, onko t tyypittyvä Γssa keräämällä pakotteet C, joiden tulee olla voimassa, jotta t:llä voisi olla tyyppi paluutyypin S kanssa ja jakamalla muuttujat C:n kanssa, joka karakterisoi mahdolliset t:n tyypit näiden muuttujien avulla. Löytääksemme ratkaisut t:lle etsimme tyyppikorvauksia σ, jotka toteuttavat C:n (eli korvaukset joilla kaikista C:n yhtälöistä tulee identiteettejä). Jokaiselle sellaiselle σ tyyppi σS on mahdollinen t:n tyyppi. Jos yhtäkään sellaista σa ei löydy, t ei ole tyypittyvä.

Määritelmä: Olkoon $\Gamma \vdash t : S \,|C$. Ratkaisu (Γ, t, S, C):lle on pari (σ, T) s.e. σ toteuttaa C:n ja $\sigma S = T$. Annetun kontekstin Γ termin t tyyppimuuttujat voidaan ilmentää kahdella tavalla; deklaratiivisesti, tai algoritmisesti tyyppipakotteiden kautta löytämällä jotkut S ja C s.e. $\Gamma \vdash t : S \,|C$.

Osoittautuu, että nämä kaksi tapaa ovat itseasiassa ekvivalentteja eli että jokainen ratkaisu (Γ, t, S, C) on myös ratkaisu (Γ, t) ja että jokainen ratkaisu (Γ, t) voidaan laajentaa ratkaisuksi (Γ, t, S, C) antamalla pakotteiden luonnin aikan generoiduille tyyppimuuttujille sopivat arvot.

4 Yhdistäminen

Voidaksemme laskea ratkaisuja pakotejoukoille käytämme yhdistämisalgoritmia, joka tarkistaa, että ratkaisujen joukko on epätyhjä ja etsii parhaimman elementin siinä mielessä, että kaikki ratkaisut voidaan suoraviivaisesti generoida siitä.

Määritelmä: Tyyppikorvaus σ on vähemmän erikoistava (tai yleisempi) kuin tyyppikorvaus σ', $\sigma \sqsubseteq \sigma'$, jos $\sigma' = \gamma \circ \sigma$ jollakin tyyppikorvauksella γ.

Määritelmä: Pääyhdistäjä (tai yleisin yhdistäjä) pakotejoukolle C on tyyppikorvaus σ joka toteuttaa C:n ja $\sigma \sqsubseteq \sigma'$ kaikille C:n toteuttaville tyyppikorvauksille σ'.

Yhdistämisalgoritmi tyypeille:

$$
\text{unify}(C) = \begin{array}{l}
if\, C \ = \emptyset, then[\,] \\
else\, let\, \{S = T\} \cup C' = C in \\
\quad if S = T \\
\qquad then\, \text{unify}(C') \\
\quad else\, if S = X\, and\, X \notin FV(T) \\
\qquad then\, \text{unify}([X \mapsto T]C') \circ [X \mapsto T] \\
\quad else\, if\, T = X\, and\, X \notin FV(S) \\
\qquad then\, \text{unify}([X \mapsto S]C') \circ [X \mapsto S] \\
\quad else\, if\, S = S_1 \to S_2\, and T = T_1 \to T_2 \\
\qquad then\, \text{unify}(C' \cup \{S_1 = T_1, S_2 = T_2\}) \\
\quad else \\
\qquad \text{fail}
\end{array}
$$

Lause "let $\{S = T\} \cup C' = C$" voidaan lukea "valitse pakote $S = T$ joukosta C ja olkoon C' joukon C jäljelle jäävät pakotteet". Riveillä viisi ja seitsemän tehtävät tarkistukset $X \notin FV(T)$ ja $X \notin FV(S)$ estävät algoritmia luomasta syklisiä korvauksia tyyliin $X \mapsto X \to X$, jollaisia ei sallita äärellisiä tyyppejä tarkastellessa. Jos kieltä laajennetaan käsittelemään myös äärettömiä ts. rekursiivisia tyyppejä, tarkistukset voidaan jättää pois. Yhdistämisalgoritmi pysähtyy aina (todistus [Pie02, s.328-329]).

5 Päätyypit

Aiemmin huomattiin, että jos on jokin tapa ilmentää jonkin termin tyyppimuuttuja siten, että termistä tulee tyypittyvä, niin silloin on olemassa yleisin tapa tehdä se.

Määritelmä: Pääratkaisu (Γ, t, S, C):lle on ratkaisu (σ, T) siten että aina kun (σ', T') on ratkaisu (Γ, t, S, C):lle, $\sigma \sqsubseteq \sigma'$ pätee. Jos (σ, T) on pääratkaisu, T on termin t päätyyppi Γssa.

Teoreema [päätyypit]: Jos (Γ, t, S, C):lle on jokin ratkaisu, sillä on myös pääratkaisu. Yhdistämisalgoritmia voidaan käyttää päättämään onko (Γ, t, S, C):llä ratkaisua ja jos on niin myös mikä pääratkaisu on.

Päätyyppejä voidaan käyttää rakentamaan tyypinpäättelyalgoritmi, joka toimii inkrementaalisemmin kuin edellä esitetty. Sen sijaan, että kaikki pakotteet generoidaan ensin ja ratkaistaan sitten, generointi ja ratkaisu voidaan lomittaa siten, että tyypinpäättelyalgoritmi palauttaa päätyypin jokaisessa vaiheessa. Koska tyypit ovat aina päätyyppejä, algoritmin ei koskaan tarvitse analysoida alitermejä uudelleen: se rajoittaa tyyppimuuttujia mahdollisimman vähän tyypittyvyyden saavuttamiseksi jokaisessa vaiheessa. Tällaisen algoritmin etuna on myös se, että se pystyy paremmin osoittamaan käyttäjälle ohjelmassa esiintyvät tyyppivirheet.

6 Implisiittiset tyyppimerkinnät

Tyypinpäättelyä tukevat ohjelmointikielet antavat ohjelmoijalle vapauden jättää osan tai kaikki tyyppimerkinnät pois lambda-abstraktioiden yhteydestä. Eräs tapa saavuttaa tämä on yksinkertaisesti pakottaa jäsentäjä täyttämään puuttuvat tiedot uusilla generoiduilla tyyppimuuttujilla. Parempi tapa on kuitenkin lisätä merkitsemättömät abstraktiot termin kielioppiin ja vastaava sääntö tyyppipakoterelaatioon:

$$\frac{X \notin \chi \quad \Gamma, x : X \vdash t_1 : T|_\chi C}{\Gamma \vdash \lambda x.t_1 : X \to T|_{\chi \cup \{X\}} C} \qquad (\mathrm{CT-ABSINF})$$

Nyt jos merkitsemättömästä abstraktiosta tehdään kopioita, CT-ABSINF –säännön avulla voidaan valita eri tyyppimuuttuja kunkin kopion argumentiksi. Muussa tapauksessa kaikki kopiot jakaisivat saman tyyppimuuttujan argumenttinaan. Asia valjennee seuraavassa kappaleessa paremmin.

7 Let-polymorfismi

Polymorfismi viittaa joukkoon kielen mekanismeja, joilla osaa ohjelmasta voidaan käyttää eri tyypeillä eri konteksteissa. Edellä esitetty tyypinpäättelyalgoritmi voidaan laajentaa tarjoamaan yksinkertaista polymorfismin muotoa nimeltä let-polymorfismi.

Let-polymorfismi muodostaa pohjan geneerisille tietorakennekirjastoille (listoille, taulukoille, puille jne.).

Tarve polymorfismille nousee seuraavanlaisista tilanteista. Oletetaan funktio double, joka soveltaa ensimmäistä argumenttiaan kaksi kertaa toiseen argumenttiinsa:

let double $= \lambda f : \text{Nat} \to \text{Nat}.\lambda a : \text{Nat}.f(f(a))$ in

double $(\lambda x : \text{Nat.succ}(\text{succ}(x))2;$

Koska haluamme soveltaa *double*a funktioon tyyppiä Nat $\to$ Nat, valitsemme tyyppimerkinnän, joka antaa sille tyypin (Nat $\to$ Nat) $\to$ (Nat $\to$ Nat). Vaihtoehtoisesti *double* voidaan määritellä siten, että sitä voidaan käyttää tuplaamaan totuusarvoinen funktio:

let double$= \lambda f : \text{Bool} \to \text{Bool}.\lambda a : \text{Bool}.f(f(a))$ in

double $(\lambda x : \text{Bool}.x)false;$

Luonnollisestikaan emme voi käyttää *samaa* double-funktiota toimimaan sekä lukujen että totuusarvojen kanssa. On siis määriteltävä kaksi versiota samasta funktiosta, jotka eroavat vain tyyppimerkintöjen osalta:

let doubleNat $= \lambda f : \text{Nat} \to \text{Nat}.\lambda a : \text{Nat}.f(f(a))$ in ...

let doubleBool $= \lambda f : \text{Bool} \to \text{Bool}.\lambda a : \text{Bool}.f(f(a))$ in ...

Edes tyyppimuuttujien käyttö tyyppimerkintöjen sijaan ei auta:

let double $= \lambda f : X \to X.\lambda a : X.f(f(a))$ in

let a = double$(\lambda x : \text{Nat.succ}(\text{succ}x))$1in

let b = double$(\lambda x : \text{Bool}.x)$false in ...

Nyt a:n määritelmä luo pakotteen $X \to X = \text{Nat} \to \text{Nat}$, mutta samaan aikaan b:n määritelmä luo pakotteen $X \to X = \text{Bool} \to \text{Bool}$, joka muuttaa koko ohjelman tyypittymättömäksi. Ongelma onkin siinä, että sama tyyppimuuttuja X on käytössä kummassakin tapauksessa. Ongelma voidaan kuitenkin korjata seuraavasti: Ensin muutetaan let'in tyypityssääntöä siten, että oikeanpuoleisen termin t_1 tyypin laskemisen ja sen käyttämisen sidotun muuttujan x tyyppinä kun lasketaan t_2:n tyyppiä

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \mathrm{let}\, x = t_1\, \mathrm{in}\, t_2 : T_2} \qquad (\mathrm{T-LET})$$

sijaan korvataan t_1 x:llä ja tarkistetaan seuraavan lausekkeen tyyppi:

$$\frac{\Gamma \vdash [x \mapsto t_1]t_2 : T_2}{\Gamma \vdash \mathrm{let}\, x = t_1 \text{ in } t_2 : T_2} \qquad (\mathrm{T-LETPOLY})$$

Samaan tapaan voidaan kirjoittaa lausekkeelle pakote:

$$\frac{\Gamma \vdash [x \mapsto t_1]t_2 : T_2 \,|_\chi\, C}{\Gamma \vdash \mathrm{let}\, x = t_1 \text{ in } t_2 : T_2 \,|_\chi\, C} \qquad (\mathrm{CT-LETPOLY})$$

Toisin sanoen let'in tyypityssääntöä muutetaan siten, että evaluoidaan let x $= v_1 int_2 \to [x \mapsto v_1]t_2$ (E-LETV) ennen tyyppien laskentaa.

Seuraavaksi kirjoitetaan *double*:n määritelmä käyttämällä implisiittistä tyyppimerkintää:

let double $= \lambda f.\lambda a.(f(f(a)))$ in

let a $= double(\lambda x : \mathrm{Nat.succ(succ}x))1$ in

let b $= double(\lambda x : \mathrm{Bool}.x)$ false in ...

Käyttämällä sääntöjä CT-LETPOLY ja CT-ABSINF saadaan kaksi kopiota *doublen* määritelmästä kun CT-ABSINF luo kummallekin abstraktiolle eri tyyppimuuttujan.

Sääntöä T-LETPOLY on vielä muutettava, koska nykyisellään abstraktion määritelmää ei tyyppitarkasteta! Korjaus on helppo tehdä lisäämällä premissi tyypityssääntöön:

$$\frac{\Gamma \vdash [x \mapsto t_1]t_2 : T_2 \qquad \Gamma \vdash t_1 : T_1}{\Gamma \vdash \mathrm{let}\, x = t_1 \text{ in } t_2 : T_2} \qquad (\mathrm{T-LETPOLY'})$$

samanlainen korjaus tehdään myös CT-LETPOLYYN.

Asiaan liittyy vielä toinenkin ongelma. Nimittäin jos let'in runko sisältää useita viittauksia sillä sidottuun muuttujaan, koko let-määritelmän oikea puoli tarkistetaan jokaisen viittauksen kohdalla erikseen huolimatta siitä, käytetäänkö implisiittistä tyyppimerkintää. Koska määritelmän oikea puoli voi itsekin sisältää let-sidontoja, kyseinen tyypityssääntö voi aiheuttaa huomattavan määrän turhaa työtä. Tällainen uudelleentarkistus voidaan kuitenkin kiertää muotoilemalla tyypityssäännöt uudelleen. termin '$letx = t1int2in\Gamma$' tyyppitarkistus voidaankin tehdä seuraavasti:

1. Käytetään pakotettuja tyypityssääntöjä ja lasketaan tyyppi S_1 ja joukko C_1 oikean puolen termille t_1.

2. Käytetään yhdistämistä, jolla löydetään yleisin ratkaisu σ rajoitteille C_1 ja sovelletaan σ:aa S_1:een (ja Γ:aan). Näin saadaan t_1:n päätyyppi T_1.

3. Yleistetään kaikki muuttujat, joita on jäljellä T_1:ssä. Jos $X_1 \ldots X_n$ ovat jäljelle jääneet muuttujat, kirjoitetaan $\forall X_1 \ldots X_n.T_1 t_1$:n päätyyppiskeemalle. Ei kuitenkaan yleistetä muuttujia T_1, jotka ovat myös Γssa.

4. Laajennetaan kontekstia sisältämään myös tyyppiskeema $\forall X_1 \ldots X_n.T_1$ sidotulle muuttujalle x ja aloitetaan t_2:n tyyppitarkistus. Yleisesti konteksti assosioi jokaisen vapaan muuttujan tyyppiskeemalla ei pelkästään tyypillä.

5. Joka kerta kun kohdataan x t_2:ssa, etsitään sen tyypiskeema $\forall X_1 \ldots X_n.T_1$. Nyt generoidaan uudet tyyppimuuttujat $Y_1 \ldots Y_n$ ja käytetääne niitä ilmentämään tyyppiskeeman, jolloin saadaan $[X_1 \mapsto Y1, \ldots, X_n \mapsto Y_n]T_1$, jota käytetään x:n tyyppinä.

Toteutettaessa let-polymorfismia johonkin ohjelmointikieleen tulee olla tarkkana sivuvaikutusten takia. Tyypityssääntöjen mukaan let-lauseke käsitellään siten, että oikeanpuoleinen termi korvataan välittömästi lausekkeen runkoon. Toisaalta evaluointisääntöjen mukaan oikeanpuoleinen termi voidaan korvata vasta kun se on redusoitu joksikin arvoksi. Joudumme siis muuttamaan joko tyypitys- tai evaluointisääntöjä. Luonnollisesti on parempi muuttaa tyypityssääntöä vastaamaan evaluointisääntöä kuin toisin päin. Käytännössä tämä voidaan toteuttaa let-sidonta polymorfisesti — s.o. sen vapaat tyyppimuuttujat voidaan yleistää — vain jos sen oikeanpuoleinen termi on syntaktinen arvo.

Lähteet

Hin97 Hindey, J.R.:Basic simple type theory, Cambridge University Press, 1997

Pie02 Pierce, B.C.:Momsen, B.:Types and programming languages, The MIT Press, 2002