Alityypityksen teoriaa

Mika Miettinen

Helsinki 19. helmikuuta 2003

Tyyppiteoria ja ohjelmointikielet -seminaari
seminaariesitelmä 20.2.2003

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Sisältö

1 Johdanto

Lähdimme liikkeelle puhtaasta, tyypittämättömästä lambdakalkyylistä ja teimme siihen sitten joukon verrattain yksinkertaisia lisäyksiä. Ensin lisäsimme tyypityksen ja Bool- ja funktiotyypit. Tämän jälkeen laajensimme kieltä tekemällä joitain yksinkertaisia lisäyksiä, kuten tietuetyypin ja viitteet. Teemme nyt käsitteellisen uudistuksen ja lisäämme kieleen *alityypityksen*, jonka idea on tuttu olio-ohjelmointikielistä.

Tässä esitelmässä käsitellään alityypityksen teoriaa Benjamin S. Piercen kirjan *Types and Programming Languages* luvun 15 [Pie02] pohjalta.

2 Alityypityksen perusteet

2.1 Johdatteleva esimerkki

Yksinkertaisesti tyypitetyn lambdakalkyylin tyyppisäännöt estävät monien hyvin käyttäytyvien ohjelmien kirjoittamisen. Esimerkiksi termi

$$(\lambda \mathsf{r} : \{\mathsf{x} : \mathsf{Nat}\}.\ \mathsf{r.x})\{\mathsf{x} = 0, \mathsf{y} = 1\}$$

ei näillä säännöillä ole hyvintyypitetty, sillä argumentin tyyppi $\{\mathsf{x} : \mathsf{Nat}, \mathsf{y} : \mathsf{Nat}\}$ ei täsmää funktion odottaman tyypin $\{\mathsf{x} : \mathsf{Nat}\}$ kanssa. Funktion lausekkeesta nähdään kuitenkin, että parametrina annetusta tietueesta oletetaan vain löytyvän kenttä x, muiden kenttien mahdollisesta olemassaolosta ei välitetä.

Alityypityksen ideana on muuttaa tyypityssääntöjä siten että myös ylläolevan kaltaiset termit olisivat hyvintyypitettyjä.

2.2 Alityyppirelaatio

Alityypityksellä on kaksi havainnolista tulkintaa. Ensimmäisen mukaan voimme pitää tyyppiä S tyypin T alityyppinä (symbolein ilmaistuna $\mathsf{S} <: \mathsf{T}$) jos jokaista tyyppiä S olevaa termiä voidaan käyttää kaikkialla missä tyypin T termiäkin. Tämä tunnetaan *turvallisen sijoituksen periaatteena* (engl.

principle of safe substitution). Toinen tapa ymmärtää relaatio $\mathsf{S} <: \mathsf{T}$ on ajatella että tyyppiä S olevien termien joukko on tyyppiä T olevien termien joukon osajoukko. Tämä periaate tunnetaan puolestaan *osajoukkosemantiikkana* (engl. *subset semantics*).

Olennaisessa asemassa jatkossa on seuraava päättelysääntö, ns, *omaksumissääntö* (engl. *rule of subsumption*).

$$\frac{\Gamma \vdash \mathsf{t} : \mathsf{S} \qquad \mathsf{S} <: \mathsf{T}}{\Gamma \vdash \mathsf{t} : \mathsf{T}} \tag{T-Sub}$$

Sääntö kertoo, että jos $\mathsf{S} <: \mathsf{T}$ ja t on tyyppiä S, on t myös tyyppiä T. Tehdään lisäksi alityyppirelaatiosta kaksi lisäoletusta. Relaation olisi oltava refleksiivinen,

$$\mathsf{S} <: \mathsf{S} \tag{S-Refl}$$

ja lisäksi transitiivinen:

$$\frac{\mathsf{S} <: \mathsf{U} \qquad \mathsf{U} <: \mathsf{T}}{\mathsf{S} <: \mathsf{T}} \tag{S-Trans}$$

Alityyppirelaatio voidaan tämän jälkeen määritellä joukkona päättelysääntöjä, joissa kaikissa johtopäätöksenä on se, että tyyppi S on tyypin T alityyppi.

2.3 Tietueet

Aloitetaan alityyppirelaation määrittely esittämällä joukko uusia päättelysääntöjä tietue- ja funktiotyyppeihin liittyen.

Esimerkissä näimme, että haluaisimme pitää tyyppiä $\mathsf{S} = \{\mathsf{l}_1 : \mathsf{T}_1, \ldots, \mathsf{l}_{n+k} : \mathsf{T}_{n+k}\}$ tyypin $\mathsf{T} = \{\mathsf{l}_1 : \mathsf{T}_1, \ldots, \mathsf{l}_n : \mathsf{T}_n\}$ alityyppinä. Voimme siis tarvittaessa "unohtaa" joukon kenttiä tietueen lopusta. Formaalisti määrittelemme seuraavan ns. *leveyssuuntaisen alityypityksen* säännön:

$$\{\mathsf{l}_i : \mathsf{T}_i{}^{i\in 1\ldots n+k}\} <: \{\mathsf{l}_i : \mathsf{T}_i{}^{i\in 1\ldots n}\} \tag{S-RcdWidth}$$

Intuitiivisesti tämän voi parhaiten ymmärtää ajattelemalla tietuemäärittelyn määrittelevän ne kentät, jotka tietueella *vähintään* on Esimerkiksi tyyppi $\{\mathsf{x} : \mathsf{Nat}\}$ määrittelisi kaikki ne tietueet jotka sisältävät vähintään tyyppiä Nat olevan kentän x.

Toiseksi voimme myös antaa tietueiden vastinkenttien tyyppien vaihdella, kunhan toinen on aina toisen alityyppi. Määritellään seuraava ns. *syvyyssuuntaisen alityypityksen* sääntö:

$$\frac{\text{kaikille } i \quad \mathsf{S}_i <: \mathsf{T}_i}{\{\mathsf{l}_i : \mathsf{S}_i{}^{i \in 1 \ldots n}\} <: \{\mathsf{l}_i : \mathsf{T}_i{}^{i \in 1 \ldots n}\}} \qquad \text{(S-RcdDepth)}$$

Lopuksi määrittelemme säännön, jonka mukaan kenttien järjestyksellä tietueessa ei ole väliä. Tämä on helposti perusteltavissa, sillä ainoan tietueille määritellyn operaation, projektion, kannalta ei kenttien järjestyksellä tietueessa ole väliä.

$$\frac{\{\mathsf{k}_j : \mathsf{S}_j^{j \in 1 \ldots n}\} \text{ on} \{\mathsf{l}_i : \mathsf{T}_i^{i \in 1 \ldots n}\} \text{: n permutaatio}}{\{\mathsf{k}_j : \mathsf{S}_j^{j \in 1 \ldots n}\} <: \{\mathsf{l}_i : \mathsf{T}_i^{i \in 1 \ldots n}\}} \qquad \text{(S-RcdPerm)}$$

Päätellään nyt näiden sääntöjen avulla, että $\{\mathsf{p} : \{\mathsf{x} : \mathsf{Bool}, \mathsf{y} : \mathsf{Nat}\}, \mathsf{z} : \mathsf{Nat}\} <: \{\mathsf{p} : \{\mathsf{y} : \mathsf{Nat}\}, \mathsf{z} : \mathsf{Nat}\}$. Seuraavassa päättelysääntöjen nimiä on lyhennetty ja permutaatiorelaatiota on merkitty merkillä $\simeq$.

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\{\mathsf{x} : \mathsf{Bool}, \mathsf{y} : \mathsf{Nat}\} \\ \simeq \{\mathsf{y} : \mathsf{Nat}, \mathsf{x} : \mathsf{Bool}\}\end{array}}{\begin{array}{c}\{\mathsf{x} : \mathsf{Bool}, \mathsf{y} : \mathsf{Nat}\} \\ <: \{\mathsf{y} : \mathsf{Nat}, \mathsf{x} : \mathsf{Bool}\}\end{array}} \text{Perm} \qquad \dfrac{}{\begin{array}{c}\{\mathsf{y} : \mathsf{Nat}, \mathsf{x} : \mathsf{Bool}\} \\ <: \{\mathsf{y} : \mathsf{Nat}\}\end{array}} \text{Width}}{\{\mathsf{x} : \mathsf{Bool}, \mathsf{y} : \mathsf{Nat}\} <: \{\mathsf{y} : \mathsf{Nat}\}} \text{Trans} \qquad \dfrac{}{\mathsf{Nat} <: \mathsf{Nat}} \text{Refl}}{\{\mathsf{p} : \{\mathsf{x} : \mathsf{Bool}, \mathsf{y} : \mathsf{Nat}\}, \mathsf{z} : \mathsf{Nat}\} <: \{\mathsf{p} : \{\mathsf{y} : \mathsf{Nat}\}, \mathsf{z} : \mathsf{Nat}\}} \text{Depth}$$

2.4 Funktiot

Kuten missä tahansa funktionaalisessa kielessä myös lambdakalkyylissä voidaan välittää funktiolle parametrina toisia funktioita. Tarvitsemme siis myös päättelysäännön alityypitykseen myös funktiotyypeille. Tämä sääntö ilmaisee milloin voimme välittää parametrina tietyntyypisen funktion kun odotetaan toisentyyppistä funktiota.

$$\frac{\mathsf{T}_1 <: \mathsf{S}_1 \qquad \mathsf{S}_2 <: \mathsf{T}_2}{\mathsf{S}_1 \to \mathsf{S}_2 <: \mathsf{T}_1 \to \mathsf{T}_2} \qquad \text{(S-Arrow)}$$

Tässä säännössä on huomattavaa, että alityyppirelaation suunta funktion argumenteille on vastakkaissuuntainen kuin funktion arvoille. Tyyppiä $\mathsf{S}_1 \to \mathsf{S}_2$ oleva funktio hyväksyy argumenteikseen tyyppiä S_1 olevia alkioita, mutta hyväksyy tietenkin myös tämän alityyppiä T_1 edustavia alkioita. Edelleen funktion tulokset ovat tyyppiä S_2, joiden voidaan myös nähdä olevan tyyppiä T_2. Siten tyyppiä $\mathsf{S}_1 \to \mathsf{S}_2$ oleva funktio voidaan myös nähdä olevan tyyppiä $\mathsf{T}_1 \to \mathsf{T}_2$.

3 Lisäpiirteitä

3.1 Top- ja Bottom-tyypit

Samalla tavalla kuin Javassa on `Object`-luokka, joka on kaikkien luokkien yliluokka, voidaan määritellä tyyppi Top siten että jokainen tyyppi on tämän alityyppi. Tarvitsemme päättelysäännön jonka mukaan jokainen tyyppi S on tyypin Top alityyppi:

$$\mathsf{S} <: \mathsf{Top} \qquad \text{(S-Top)}$$

Voisimme myös määritellä *minimaalisen* tyypin Bot, joka on kaikkien muiden tyyppien alityyppi:

$$\mathsf{Bot} <: \mathsf{S} \qquad \text{(S-Bot)}$$

Tästä ominaisuudesta johtuu, että tyyppi Bot on tyhjä eli yksikään termi ei voi olla tätä tyyppiä. Jos nimittäin jollakin termillä v olisi tyyppi Bot, voisimme päätellä esim. $\vdash \mathsf{v} : \{\}$, mutta myös $\mathsf{v} : \mathsf{Top} \to \mathsf{Top}$.

3.2 Tyypinmuutos

Olemme tähän asti käsittäneet tyypinjulistuksen (ascription) lähinnä ohjelmakoodin dokumentaationa: ohjelmoija on piirteen avulla voinut lisätä ohjelmaan oletuksen, että seuraavassa termi on jotain annettua tyyppiä.

Alityypityksen mukaantultua tyypinjulistuksen merkitys muuttuu ja siitä tulee tavanomaisista ohjelmointikielistä tuttu tyypinmuutos. Tyypinmuutokset ylöspäin (missä termille annetaan sille "luonnostaan" kuuluvan tyypin ylätyyppi) ovat suoraviivaisia. Aiemmasta tuttu tyypinmuutossääntö säilyy ennallaan:

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}}{\Gamma \vdash \mathsf{t}_1 \ \mathsf{as}\ \mathsf{T} : \mathsf{T}} \qquad \text{(T-ASCRIBE)}$$

Tyypinmuutos alaspäin on monimutkaisempi. Tehdään ensin seuraava sääntö:

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{S}}{\Gamma \vdash \mathsf{t}_1 \ \mathsf{as}\ \mathsf{T} : \mathsf{T}} \qquad \text{(T-DOWNCAST)}$$

Oletettavasti ohjelmoija tällaisissa tapauksissa tietäisi, että termi johon sääntöä sovelletaan olisi todellisuudessa tyyppiä T. Tällä säännöllä on sellainen vaikutus, että tyypinpäättelijä ensin tarkastaisi että termillä t_1 on jokin tyyppi S ja sitten antaisi termille tyypin T. Kielen turvallisuuden kannalta tällainen sääntö on tietenkin huono, sillä tyyppien S ja T suhteesta ei oleteta mitään. Parempaan ei kuitenkaan käännösaikana ole mahdollista päästä. Tämä voidaan ratkaista tarkastamalla ajoaikana että termi johon tyypinmuutosta sovellettiin todella on tyyppiä T:

$$\frac{\vdash \mathsf{v}_1 : \mathsf{T}}{\mathsf{v}_1 \ \mathsf{as}\ \mathsf{T} \to \mathsf{v}_1} \qquad \text{(E-DOWNCAST)}$$

3.3 Variantit

Alityypityssäännöt varianteille ovat hyvin samankaltaiset kuin tietueille. Ainoa muutos tapahtuu leveyssuuntaisen alityypityksen säännössä, joka nyt antaa mahdollisuuden *lisätä* uusia kenttiä siirryttäessä tyyppihierarkiassa ylöspäin:

$$< \mathsf{l}_i : \mathsf{T}_i^{i\in 1\ldots n} > <: < \mathsf{l}_i : \mathsf{T}_i^{i\in 1\ldots n+k} > \qquad \text{(S-VARIANTWIDTH)}$$

Alityypityksen mukaantultua voimme tässä yhteydessä luopua liputuskonstruktiosta. Määrittelemme nyt variantin tyypityssäännön seuraavasti:

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{T}_1}{\Gamma \vdash < \mathsf{l}_1 = \mathsf{T}_1 > : < \mathsf{l}_1 : \mathsf{T}_1 >} \qquad \text{(T-VARIANT)}$$

Liputus oli alunperin mukana tyyppien yksikäsitteisyyden takia, nyt voidaan termille $< \mathsf{l} = \mathsf{t} >$ antaa sen tarkka tyyppi; muut tyypit saadaan tästä alityypityksellä.

3.4 Viitteet

Viitteille voisimme määritellä seuraavan säännön:

$$\frac{\mathsf{S}_1 <: \mathsf{T}_1 \qquad \mathsf{T}_1 <: \mathsf{S}_1}{\mathsf{Ref}\ \mathsf{S}_1 <: \mathsf{Ref}\ \mathsf{T}_1} \tag{S-Ref}$$

Tämä sääntö on selvästikin hyvin rajoittava, mikä johtuu siitä että viitettä voidaan käyttää sekä lukemiseen että kirjoittamiseen.

Parempaan päästään esittelemällä kaksi uutta viitetyyppiä: luku- ja kirjoitusviitteet. Lukuviite $\mathsf{Source}\ \mathsf{T}$ on viite, jota voidaan käyttää vain tyypin T alkioiden lukemiseen. Kirjoitusviite $\mathsf{Sink}\ \mathsf{T}$ on taas viite, jota voidaan käyttää tyyppiä T olevien alkioiden kirjoittamiseen.

Lukemisen ja kirjoittamisen tyypityssäännöt saavat nyt seuraavan muodon.

$$\frac{\Gamma \mid \Sigma \vdash \mathsf{t}_1 : \mathsf{Source}\ \mathsf{T}_{11}}{\Gamma \mid \Sigma \vdash !\mathsf{t}_1 : \mathsf{T}_{11}} \tag{T-Deref}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathsf{t}_1 : \mathsf{Sink}\ \mathsf{T}_{11} \qquad \Gamma \mid \Sigma \vdash \mathsf{t}_2 : \mathsf{T}_{11}}{\Gamma \mid \Sigma \vdash \mathsf{t}_1 := \mathsf{t}_2 : \mathsf{Unit}} \tag{T-Assign}$$

Lukuviitettä koskee nyt seuraava alityypityssääntö:

$$\frac{\mathsf{S}_1 <: \mathsf{T}_1}{\mathsf{Source}\ \mathsf{S}_1 <: \mathsf{Source}\ \mathsf{T}_1} \tag{S-Source}$$

Kirjoitusviitettä koskee taas sääntö

$$\frac{\mathsf{T}_1 <: \mathsf{S}_1}{\mathsf{Sink}\ \mathsf{S}_1 <: \mathsf{Sink}\ \mathsf{T}_1} \tag{S-Sink}$$

Tavallinen viite $\mathsf{Ref}\ \mathsf{T}_1$, josta voi lukea ja johon voi kirjoittaa, on luku- ja kirjoitusviitteiden yhdistelmä. Esitellään lopuksi kaksi sääntöä, jotka kertovat tämän.

$$\mathsf{Ref}\ \mathsf{T}_1 <: \mathsf{Source}\ \mathsf{T}_1 \tag{S-RefSource}$$

$$\mathsf{Ref}\ \mathsf{T}_1 <: \mathsf{Sink}\ \mathsf{T}_1 \tag{S-RefSink}$$

Lähteet

Pie02 Pierce, B.C., Types and Programming Languages. *The MIT Press, Cambridge, Massachusetts, 2002.*