Annika Piiroinen, Kalle Viiri 2014

Johdatus λ-kalkyyliin

λ-kalkyyli on alunperin Alonzo Chruchin kehittämä Turing-täydellinen formaalin laskennan malli. Funktionaaliset ohjelmointikielet perustuvat siihen, ja sillä on sovelluksia myös matematiikan, lingvistiikan ja filosofian aloilla.

Käymme tässä esseessä läpi aluksi yleisen johdatuksen ideamaailmaan λ-kalkyylin takana, minkä jälkeen esitämme λ-kalkyylin formaalin määritelmän selittävien esimerkkien kera. Sovellamme annettuja määritelmiä yksinkertaisen kaksiarvologiikan parissa, minkä jälkeen lukijalla on toivottavasti alustava käsitys λ-kalkyylin perustoiminnasta ja hän on valmis siirtymään haastavampien aihetta käsittelevien tekstien pariin.

Johdatus λ-ajatteluun

Niin kuin kaikki laskennan mallit, kuten Turingin kone, myös λ-kalkyyli esittää maailman omanlaisessaan, yksinkertaistetussa muodossa, joka ei ulkoisesti muistuta alkuperäistä kohdettaan. Esimerkiksi Turingin koneessa numerot saatetaan esittää jonona ykkösiä nauhalla, kun taas λ-kalkyylissä ne esitetään tietynmuotoisina funktioina. Olennaista kuitenkin on, että mallin sisäisiä laskusääntöjä noudatettaessa nämä numerot käyttäytyvät niin kuin vastineensa. “Jos se haisee numerolta kolme ja maistuu numerolta kolme, se on numero kolme.”

λ-kalkyylissä kaikki asiat esitetään jonkinlaisina funktioina. Nyt ei kannata kuitenkaan ajatella funktioita esimerkiksi kuvauksina lukujen joukolta toiselle, vaan yleensä asioina, jotka muuttavat saamansa syötteen johonkin toiseen muotoon. Lisäksi, koska kaikki λ-kalkyylin maailman objektit ovat funktioita, voivat funktiot olla kuvauksia vain näiden funktioiden joukolta samaiselle funktioiden joukolle.

Turingin koneessa koko maailma kuvataan merkkeinä nauhalla ja maailman tapahtumat koneen toimintana, kun taas λ-kalkyylissä maailman asiat ovat funktioita ja tapahtumat sitä, kun nämä funktiot ottavat sisäänsä toisia funktioita ja tekevät niistä joitain toisia funktioita.

Miten sitten voi kuvata funktioita ilman referenssejä mihinkään muuhun kuin toisiin funktioihin? Meillä on olemassa käsitteitä, kuten identiteettikuvaus, vakiokuvaus ja yhdistetty kuvaus, jotka riippuvat vain funktion käsitteestä itsestään. Ne riittävät tarpeeksi useilla eri tavoilla yhdessä sovellettuina muodostamaan rikkaan maailman, josta löytyy vastine kaikille niille olennoille, joiden toimintaa haluamme mallintaa.

λ-kalkyylin maailma rakentuu

Seuraavaksi määrittelemme, minkä muotoisia ovat ne objektit, jotka kuuluvat λ-kalkyylin maailmaan. Näitä objekteja kutsutaan *λ-termeiksi*. Formaalia määritelmää selitetään auki pian sen esittämisen jälkeen.

λ-termin määritelmä

1. Muuttuja a itsessään on λ-termi.
2. Jos t on λ-termi ja x muuttuja, niin (λx.t) on λ-termi. (*abstraktio*)
3. Jos t ja s ovat λ-termejä, niin (ts) on λ-termi. (*sovellus*)

Määritelmän ensimmäinen kohta on induktiivisen määritelmämme ensimmäinen askel. Se kertoo, että voimme merkitä λ-kalkyylin funktiota yleensä jollain kirjaimella. Tällainen muuttuja a voi siis edustaa mitä tahansa funktiota. Vain funktion esitetty toiminta määrittää sen, mikä funktio se on. Siten siis näitä muuttujasymboleja voi vaihtaa, esimerkiksi a:n sijaan voisimme yhtä hyvin käyttää b:tä ja niin edelleen. Symboleita ei toki saa vaihtaa siten, että merkitys muuttuu.

Määritelmän kohta kaksi kertoo funktion toiminnasta. Se on kuin funktion toiminta kirjoitettuna auki. Merkintä (λx.t) tarkoittaa, että λ-symbolin ja pisteen välillä oleva asia, tässä tapauksessa siis x, kuvautuu pisteen jälkeen annettuun muotoon, tässä siis muotoon t. Luultavasti termi t sisältää symbolin x, mutta ei välttämättä.

Konkretisoidaksemme määritelmää hieman, ajatelkaamme hetken “perinteistä” funktiota, esimerkiksi funktiota $x \mapsto x+2$. Tämän funktion voisi esittää λ-kalkyylin hengessä muodossa (λx.x+2). Tässä termiä t vastaisi siis lauseke x+2. λ-kalkyylissä ei kuitenkaan ole olemassa symbolia + tai 2, mutta termi t voisi pitää sisällään symbolin x jossain muussa muodossa.

Määritelmän kohdan kaksi muotoista λ-termiä kutsutaan *abstraktioksi*, koska se esittää funktion abstraktissa muodossa. Termin uloimmat sulut voi jättää pois tapauksissa, joissa niillä ei ole merkityksellistä arvoa, esimerkiksi sellaisenaan λx.t on sama asia kuin (λx.t). Kun λ-termejä kirjoitetaan sisäkkäin, sulut kuitenkin ovat usein tarpeelliset erottimet.

Kolmas kohta määritelmästä vastaa “perinteisessä funktioajattelussa” sitä, että funktiolle annetaan jokin syöte. Konkretisoidakemme ajatusta, käytämme taas λ-maailman ulkopuolista esimerkkiä. Olkoon siis termin (ts) termi t = (λx.x+2) ja s=4. Nyt merkintä (ts) eli ((λx.x+2) 4) tarkoittaisi sitä, että funktiolle $x \mapsto x+2$ annetaan syötteeksi luku 4. Kun lauseke evaluoidaan, saadaan arvoksi 4+2 = 6. Toisin sanoen ((λx.x+2) 4) = 6. Edelleenkään λ-kalkyylin sallittuihin merkkeihin ei kuulu numeroita tai yhteenlaskua, mutta λ-termien evaluoiminen tapahtuu samalla tavalla kuin annetussa esimerkissä.

((λx.x+2) 4)

Kun kaksi λ-termiä kirjoitetaan peräkkäin, kuten yllä, saatu uusi λ-termi voidaan evaluoida ottamalla ensimmäisen termin pisteen jälkeinen osa (tässä x+2) ja sijoittamalla siihen pistettä edeltävän symbolin (x) paikalle jälkimmäinen λ-termi (4). (Saatiin siis 4+2). Voit ajatella tätä toimenpidettä eräänlaisena leikkaa ja liitä -operaationa. Tähän evaluointiin palataan vielä useampien esimerkkien kera.

Määritelmän kolmannen kohdan mukaista λ-termiä kutsutaan sovellukseksi, koska siinä *ensimmäistä λ-termiä sovelletaan jälkimmäiseen λ-termiin*.

Logiikkaa

Ehkä helpoimman esimerkin λ-kalkyylin pariin tarjoaa kaksiarvologiikka. Arvot TRUE ja FALSE on määritelty tietynlaisina yksinkertaisina funktioina, ja konnektiivit sopivanlaisina funktioina siten, että kun niitä soveltaa totuusarvoihin, ne palauttavat sen totuusarvon, mikä pitääkin.

Totuusarvot λ-kalkyylissä
TRUE ≔ (λx.(λy.x)) ja
FALSE ≔ (λx.(λy.y)).

Kun tarkastelemme määritelmiä, huomaamme, että TRUE on siis funktio, joka palauttaa vakiofunktion, jonka vakio on ensiksi annettu arvo. FALSE puolestaan on funktio, joka palauttaa aina identiteettikuvauksen.

Tässä vaiheessa lukijakin ehkä huomaa, että nykyisellä tavalla λ-termien induktiivista määritelmää sovellettaessa sulkuja alkaa kertyä melko paljon sisäkkäin. Siksi sovimmekin seuraavanlaisen merkintäsäännön:

$$(\lambda x.(\lambda y.z)) \Leftrightarrow (\lambda xy.z)$$

On tärkeää huomata, että nyt siis (λxy.z)(q) = (λy.z) [x≔q], missä merkintä (λy.z) [x≔q] tarkoittaa sitä, että otetaan termi (λy.z) mutta korvataan siitä kaikki x-symbolit q-termillä.
Koska tämän lyhennysmerkinnän käytön hallitseminen ja ymmärtäminen on olennaista myöhemmän seuraamisen kannalta, kannattanee lukijan tässä välissä halutessaan käydä läpi äsken esitetty muunnos. Esimerkiksi lukija voi vaikka evaluoida seuraavan termin, joka muiden konkretisoivien esimerkkiemme tavoin käyttää “ei-sallittua” +-merkkiä: (λxy.x+y)(4). Tulokseksi pitäisi saada (λy.4+y). Kiinnitettäköön huomiota siihen, kuinka muuttuja x “tippuu” lausekkeen edestä ja luku 4 “menee sen sisään”.

Lyhennysmerkintää käyttäen totuusarvot muuttuvat seuraavaan muotoon:
TRUE = (λxy.x) ja

FALSE = (λxy.y).

Määrittelemme sulkujenkarsimissyistä lisäksi, että
abc = (ab)c, mutta
λa.bc = (λa.bc).

Tähän mennessä emme ole lainkaan puuttuneet siihen, voiko λ-kalkyylissä olla useamman muuttujan funktioita. Tavallaan kysymys on epäolennainen, sillä Curry-muunnoksen avulla kaikki useamman muuttujan funktiot voidaan esittää yhdistettyinä yhden muuttujan funktioina. λ-kalkyylissäkin siis esimerkiksi muotoa λab.c -olevat funktiot voidaan tulkita kahden muuttujan funktioiksi, ja niin usein kannattaakin tehdä, sillä se helpottaa joidenkin funktioiden toiminnan ymmärtämistä.

NOT-funktio

Jotta funktio NOT käyttäytyisi niin kuin haluamme sen käyttäytyvän, sen on oltava muotoa
NOT = (λpab.pba).

Nyt
NOT TRUE =
(λpab.pba)(λxy.x) = (λab.(λxy.x)ba) = (λab.(λy.b)a) = (λab.b) = FALSE,
kuten pitääkin. Muista, että symbolien valinta ei vaikuta funktion toimintaan, joten (λxy.y) = (λab.b).

Edellä olevaa oli varmasti hankalaa seurata! λ-termien evaluoiminen “käsin” on tuskallinen tehtävä etenkin, kun lausekkeet pitenevät. Evaluoimisen automatisoiminen (tietokoneohjelman kirjoittaminen) ja erilaisten apukeinojen käyttäminen käyvät helposti mielessä. Jos λ-kalkyylin toimintaa haluaa ymmärtää, on hyvä kuitenkin pyöritellä pari laskutoimitusta itse paperilla. Onneksi valmiiksi laskettujen laskujen seuraaminen helpottuu, kun ne värikoodaa.

Tässä on merkitty purppuralla sen termin sulut ja λ-symboli, jota sovelletaan johonkin. Sinisellä on merkitty korvattava symboli, keltaisella lauseke, johon sijoitetaan sovelluksen kohde, ja vihreällä se, mihin λ-termiä sovelletaan.

Esimerkiksi sovellus (λx.t) s väritetään muotoon (λx.t) s.

1. (λpab.pba)(λxy.x) =
2. (λab.(λxy.x)ba) =
3. (λab.(λy.b)a) =

4. (λab.b)

- Kohdassa yksi vihreällä merkitty termi sijoitetaan keltaiseen lausekkeeseen niihin kohtiin, missä sinisellä merkitty symboli, eli p, ilmenee. Sinisellä merkitty termi katoaa.

- Kohdassa kaksi ensimmäisen pisteen jälkeen on kolme λ-termiä peräkkäin: (λxy.x), b ja a. Aiemman sopimuksen mukaan tulkitaan abc = (ab)c, minkä vuoksi termiä (λxy.x) sovelletaan vain termiin b.

- Kolmannessa kohdassa vihreällä merkitty termi, a, sijoitetaan keltaiseen lausekkeeseen niihin kohtiin, missä sinisellä merkitty symboli, y, ilmenee. Symboli y ei kuitenkaan ilmene keltaisessa lausekkeessa, joten a “katoaa”, jättäen jäljelle vain b:n.

Kun λ-termejä evaluoi, ei kannata välttämättä ajatella funktioita ja niiden kuvauksia. Yksinkertainen tapa hahmottaa λ-kalkyylissä tapahtuvat symbolien muokkaukset on ymmärtää ne “leikkaa ja liitä” -operaatioina. Tässä vihreä termi leikattiin ja liitettiin keltaiseen lausekkeeseen niihin kohtiin, joissa sininen symboli esiintyy.

Analysoidaan vielä funktioiden TRUE, FALSE ja NOT toimintaa. FALSE (λxy.y) on kahden muuttujan funktio, joka “tiputtaa” ensimmäisen syötteen pois ja jättää jälkimmäisen. TRUE (λxy.x) puolestaan “säästää” ensimmäisen, mutta “tiputtaa” jälkimmäisen.

NOT (λpab.pba) -funktio kannattaa ajatella yhden muuttujan funktiona, joka palauttaa kahden muuttujan funktion. Se asettaa syötteensä p termien b ja a eteen. Jos p on FALSE, b “tipahtaa” ja a “säilyy”, eli saadaan TRUE, (λab.a). Jos taas p on TRUE, b “säilyy” ja a “tipahtaa”.

Esittelemme vielä lyhyesti loogiset operaatiot AND ja OR, eli konjunktio ja disjunktio:

AND = (λab.ab(λcd.d)) = (λab.ab FALSE)
OR = (λab.a(λcd.c)b) = (λab.a TRUE b)

Konjunktiomme toimii siis niin, että ensimmäisen operandin ollessa (λab.a) eli TRUE, AND evaluoituu toisen operandin arvoksi. Muussa tapauksessa se evaluoituu suoraan FALSEksi. Disjunktio vastaavasti evaluoituu TRUEksi jos ensimmäinen operandi on TRUE, muussa tapauksessa toisen operandin arvoksi.

Muu λ-maailma

Lukija voi halutessaan itse ottaa selvää muista λ-kalkyylin logiikkafunktioista tai päätellä itse, mitä niiden tulee olla, jotta ne toimisivat halutulla tavalla. Kirjoittajat eivät tarkoita ehdottaa, ettäkö tämä

olisi helppo tehtävä. Huomautettakoon, että kahden argumentin funktiot, kuten AND ja OR, toimivat prefix-muodossa, esimerkiksi kirjoitetaan ennemmin “AND x y” kuin “x AND y”.

Luvut esitetään λ-kalkyylissä Peanon aksioomien henkisesti. Määritellään
0 ≔ (λ ab.b),
1 ≔ (λab.b(a)),
2 ≔ (λab.b(b(a))),
3 ≔ (λab.b(b(b(a)))) jne.

Laskutoimitukset määritellään näiden mukaan. Sanottakoon yleisesti, että muodostuvat funktiot ovat melko elegantteja matemaattisesta näkökulmasta. Lukujen koodaustyylin yksi etu on, että lukua kuvaavan funktion soveltaminen toiseen funktioon johtaa toisen funktion “monistumiseen”: näin ollen λ-maailmassa voidaan suorittaa funktio *Z x* kertaa vain laittamalla ne peräkkäin *xZ*.

Rekursio

Rekursio tarkoittaa yleisesti funktion soveltamista itseensä. λ-kalkyyli tunnetaan runsaasta rekursion käytöstä, sillä se siinä ei ole muuta rakennetta laskutoimitusten mielivaltaiselle toistamiselle (vertaa: silmukat ohjelmointikielissä).

Rekursiivinen funktio λ-kalkyylissa voi näyttää esimerkiksi seuraavalta:

(λa(λb.a(bb))(λb.a(bb)) **F**

(missä **F** olkoo jokin mielivaltainen funktio)

Evaluoimalla λ-lauseketta saamme:

(λb.**F**(bb))(λb.**F**(bb))
F((λb.**F**(bb))(λb.**F**(bb)))

Kutsukaamme funktiota *(λa(λb.a(bb))(λb.a(bb))* nimellä **G**. Nyt näimme että **G F = F(G F)**. Tällä rekursiolla ei ole pohjatapausta, mutta käyttämällä loogisia funktioita voidaan sellainenkin määritellä sopivan toiston aikaansaamiseksi.